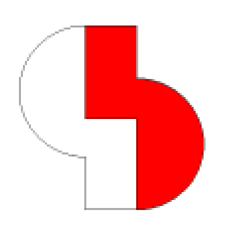


Bartels User Language®
Programmierhandbuch

Bartels User Language Programmierhandbuch
Herausgeber: Bartels System GmbH, München
Stand: November 2013

Vorwort

Das Bartels User Language - Programmierhandbuch enthält die Beschreibung der **Bartels User Language**-Programmiersprache sowie ausführliche Informationen über die Art der Einbindung sowie die Möglichkeiten der Anwendung im **Bartels AutoEngineer**-EDA-System. Im Bartels User Language - Programmierhandbuch finden Sie detaillierte Informationen zu den folgenden Themenschwerpunkten:

- Grundkonzept und Sprachbeschreibung der **Bartels User Language**
- das **Bartels User Language**-Programmiersystem: **User Language Compiler** und **User Language Interpreter**
- **User Language**-Beispielprogramme; Informationen zu den mit dem **Bartels AutoEngineer** ausgelieferten **User Language**-Programmen
- Beschreibung der Datentypen für den Zugriff auf die Designdaten im **Bartels AutoEngineer**
- Beschreibung der in der **Bartels User Language** implementierten Systemfunktionen

Der Leser sollte vertraut sein mit der Benutzung seines Betriebssystems und mit der Handhabung eines auf seinem System verfügbaren Editors zur Erstellung von ASCII-Dateien. Darüber hinaus sollte der Anwender über hinreichende Erfahrungen in der Benutzung des **Bartels AutoEngineer** verfügen. Schließlich werden Kenntnisse der Programmierung im Allgemeinen sowie der Programmiersprache C im Speziellen vorausgesetzt.

AutoEngineer-Anwender, die keine eigene **User Language**-Programmierung planen, sollten zumindest einen Blick auf Kapitel 4 werfen, da darin Kurzbeschreibungen der mit der BAE-Software ausgelieferten **User Language**-Programme aufgelistet sind.

Beachten Sie bitte vor Verwendung der in dieser Dokumentation enthaltenen Informationen und der darin beschrieben Produkte die Copyright-Hinweise. Der Leser sollte darüber hinaus auch mit den in dieser Dokumentation verwendeten Begriffen und Konventionen vertraut sein.

Gliederung

Kapitel 1	enthält einleitende Anmerkungen zur **Bartels User Language**.
Kapitel 2	enthält die Sprachbeschreibung der **Bartels User Language**.
Kapitel 3	beschreibt das Programmiersystem der **Bartels User Language**, also den **Bartels User Language Compiler** und den **Bartels User Language Interpreter**.
Kapitel 4	enthält eine Übersicht über die mit dem **Bartels AutoEngineer** ausgelieferten **User Language**-Includedateien und **User Language**-Programme sowie Hinweise zur Bereitstellung der Programme im **Bartels AutoEngineer**.
Anhang A	beschreibt die Konventionen für den Zugriff auf die in der **Bartels User Language** definierten Index-Variablen-Typen und Systemfunktionen sowie die hierfür definierten Wertebereiche.
Anhang B	beschreibt die in der **Bartels User Language** per Definition festgelegten Index-Variablen-Typen.
Anhang C	beschreibt die in der **Bartels User Language** eingebundenen Systemfunktionen.

Weitere Dokumentation

Die Bartels AutoEngineer® - Installationsanleitung beschreibt die Konfigurationen und Systemvoraussetzungen des **Bartels AutoEngineer** und enthält detaillierte Anweisungen zur Installation des **Bartels AutoEngineer** auf unterschiedlichen Hardware- und Softwareplattformen.

Das Bartels AutoEngineer® - Benutzerhandbuch enthält die Bedienungsanleitung für das **Bartels AutoEngineer**-CAE/CAD/CAM-System. Im Bartels AutoEngineer® - Benutzerhandbuch finden Sie detaillierte Informationen zu den folgenden Themenschwerpunkten:

- Einleitung: Systemarchitektur, allgemeine Bedienungshinweise, Designdatenbank
- Schaltungsentwurf (CAE), **Schematic Editor**
- Netzlistenverarbeitung, Forward- und Backward-Annotation
- Leiterplattenentwurf (CAD) und Fertigungsdatenerzeugung (CAM), **Layouteditor** für grafisch-interaktives Leiterkartenlayout, **Autoplacement**, Flächenautomatik, Autorouting, **CAM-Prozessor**, **CAM-View**
- IC-/ASIC-Design, **Chipeditor** für grafisch-interaktives IC-Maskenlayout, **Cellplacer** und **Cellrouter** für "Place & Route", Import- und Export von GDS- und CIF-Daten
- Neuronales Regelsystem
- Utilityprogramme

Die Bartels AutoEngineer® - Symbol- und Bauteilbibliotheken Dokumentation enthält detaillierte Informationen zu den mit dem **Bartels AutoEngineer**-CAE/CAD/CAM-System ausgelieferten Symbol- und Bauteilbibliotheken.

Wünsche, Anregungen, Fragen, Probleme

Für Hinweise auf Fehler sowie Wünsche und Anregungen in Bezug auf die Implementierung neuer oder die Weiterentwicklung bestehender Funktionen bzw. Programmteile des **Bartels AutoEngineer** bzw. der **Bartels User Language** sind wir Ihnen dankbar. Sollten Sie Fragen zur **Bartels User Language** haben, oder Probleme bei deren Benutzung auftreten, so wenden Sie sich bitte an unsere Support-Abteilung. Unsere Anschrift können Sie der Bartels Website unter http://www.bartels.de entnehmen.

Begriffe

Der Leser sollte vertraut sein mit den folgenden in der **Bartels AutoEngineer**-Dokumentation immer wiederkehrenden Begriffen:

Maus	Zeigegerät (Maus, Trackball, etc.) zum Bewegen des Menübalkens und des Fadenkreuzes sowie zur Aktivierung von Funktionen verwendet
Info-Feld	Feld zur Anzeige von System-Statusmeldungen rechts oben am Bildschirm
Hauptmenü	Fest vorgegebene Funktionsauswahl im oberen Bereich der rechten Bildschirmseite zur Selektion eines Menüs
Menü	Über Hauptmenü eingestellte Funktionsauswahl im unteren Bereich der rechten Bildschirmseite
Untermenü	Aus Menüfunktion aufgerufene, untergeordnete Funktionsauswahl im unteren Bereich der rechten Bildschirmseite
Grafikarbeitsbereich	Arbeitsbereich für Grafik-Interaktionen im linken, oberen Bildschirmbereich
Status-Zeile	Unterste Bildschirmzeile zur Anzeige von System-Statusmeldungen bzw. für Benutzerabfragen
Menübalken	Menükursor zum Anwählen einer Menüfunktion
Fadenkreuz	Kursor im Grafikarbeitsbereich
Menüprompt	Benutzerabfrage in der Status-Zeile
Popupmenü	Optional über dem Grafikarbeitsbereich angezeigtes Auswahlmenü für funktionsspezifische Objekte, Elemente oder Arbeitsschritte
Button	Selektierbarer Popupmenüeintrag zur Anwahl eines speziellen Elements oder zur Aktivierung einer menüspezifischen Funktion
Funktion anwählen	Den Menübalken mittels Maus auf die Menüfunktion positionieren
Aktivieren	Betätigen der Maustaste
Pick	Ein zu manipulierendes Element mit dem Grafikkursor selektieren
Positionieren (auch: **Place**)	Ein Element mittels Grafikkursor im Grafikarbeitsbereich positionieren
Selektieren	Auswählen eines zu bearbeitenden Elements oder einer Funktion durch Betätigen der Maustaste
Bestätigen	Die Ausführung einer über Benutzerabfrage verifizierten Funktion veranlassen

In der **Bartels AutoEngineer**-Dokumentation werden die folgenden Akronyme verwendet:

BAE	Akronym zur Identifikation der **Bartels AutoEngineer**-EDA-Software
BAEICD	Akronym für das **Bartels AutoEngineer**-IC/ASIC-Designsystem, welches optional in workstationbasierenden BAE-Konfigurationen enthalten ist
SCM	Akronym für das **Schematic Editor**-Programm-Modul für den Schaltungsentwurf und zur Stromlaufplanerfassung im **Bartels AutoEngineer**
GED	Akronym für den grafischen **Layouteditor** (**Layouteditor**) des **Bartels AutoEngineer** PCB-Designsystems
AR	Akronym für den **Autorouter** des **Bartels AutoEngineer** PCB-Designsystems
NAR	Akronym für den **Neuronalen Autorouter** des **Bartels AutoEngineer** PCB-Designsystems
CAM	Akronym für den **CAM-Prozessor** des **Bartels AutoEngineer** PCB-Designsystems
CV	Akronym für das **CAM-View**-Programm-Modul des **Bartels AutoEngineer** PCB-Designsystems
CED	Akronym für das **Chip Editor**-Programm-Modul des **Bartels AutoEngineer** IC/ASIC-Designsystems
CP	Akronym für das **IC-Autoplacement**-Programm-Modul des **Bartels AutoEngineer** IC/ASIC-Designsystems
CR	Akronym für das **IC-Autorouter**-Programm-Modul des **Bartels AutoEngineer** IC/ASIC-Designsystems
UL	Akronym für die **Bartels User Language**-Programmiersprache
ULC	Akronym für den **Bartels User Language Compiler**
ULI	Akronym für den **Bartels User Language Interpreter**

Konventionen

Soweit nicht anders vermerkt sind in der **Bartels AutoEngineer**-Dokumentation folgende symbolische Konventionen relevant:

<table>
<tr><td>Lineprint</td><td>Schreibmaschinenzeichensatz kennzeichnet durch das System ausgegebenen Text.</td></tr>
<tr><td>Boldface</td><td>Fett gedruckte Worte oder Zeichen in Format- oder Kommandobeschreibungen kennzeichnen feststehende Begriffe oder syntaktische Terminalzeichensequenzen, also direkt einzusetzende Kommando- oder Schlüsselwörter.</td></tr>
<tr><td>Emphasize</td><td>Emphatische Textauszeichnung dient der optischen Hervorhebung.</td></tr>
<tr><td>" "</td><td>Anführungszeichen dienen der Kennzeichnung von (Pfad-)Namen oder spezifizieren direkt einzugebende Zeichen(sequenzen).</td></tr>
<tr><td>[ ]</td><td>Eckige Klammern in Format- oder Kommandobeschreibungen umschließen wahlweise angebbare Elemente.</td></tr>
<tr><td>{ }</td><td>Geschweifte Klammern in Format- oder Kommandobeschreibungen umschließen eine Liste von Elementen, aus denen eines anzugeben ist.</td></tr>
<tr><td>|</td><td>Ein vertikaler Strich trennt Elemente aus einer Liste wahlweise angebbarer Elemente.</td></tr>
<tr><td>< ></td><td>Gewinkelte Klammern umschließen den logischen Namen einer (zu betätigenden) Taste oder eine semantisch zu ersetzende syntaktische Variable in einer Format- oder Kommandobeschreibung.</td></tr>
<tr><td>></td><td>Fett gedruckte Größerzeichen innerhalb Schreibmaschinenzeichensatz kennzeichnen Eingabeaufforderungen auf Betriebssystemebene.</td></tr>
<tr><td>...</td><td>Horizontale Auslassungspunkte kennzeichnen die wahlweise Wiederholbarkeit des vorhergehenden Elements in einer Format- oder Kommandobeschreibung oder die Auslassung irrelevanter Teile eines Beispieles oder einer Abbildung.</td></tr>
<tr><td>:</td><td>Vertikale Auslassungspunkte kennzeichnen die Auslassung irrelevanter Teile einer Abbildung, eines Beispieles, oder einer Format- oder Kommandobeschreibung.</td></tr>
<tr><td></td><td>beliebige Maustaste (MB)</td></tr>
<tr><td></td><td>Linke Maustaste (LMB)</td></tr>
<tr><td></td><td>Mittlere Maustaste (MMB)</td></tr>
<tr><td></td><td>Rechte Maustaste (RMB)</td></tr>
<tr><td></td><td>Tastatur(eingabe) - Return-/Eingabetaste (CR)</td></tr>
<tr><td>a b ...</td><td>Tastatur(eingabe) - Standardtaste(n)</td></tr>
<tr><td>F1 F2 ...</td><td>Tastatur(eingabe) - Funktionstaste(n)</td></tr>
<tr><td>filename</td><td>Datei- bzw. Verzeichnispfadname.</td></tr>
<tr><td>keyword</td><td>Syntaktische Terminalzeichensequenz, also direkt einzusetzendes Kommando bzw. Schlüsselwort.</td></tr>
<tr><td>message</td><td>Von BAE oder System angezeigte Status- oder Fehlermeldung.</td></tr>
<tr><td>Menu</td><td>Bartels AutoEngineer Menü.</td></tr>
<tr><td>Menu Function</td><td>Bartels AutoEngineer Menüfunktion.</td></tr>
<tr><td>Menu Option</td><td>Bartels AutoEngineer Menüoption.</td></tr>
<tr><td>Button</td><td>Bartels AutoEngineer (Popup-)Menübutton.</td></tr>
<tr><td>ul.ulc</td><td>(Hypertextlink zu) Bartels User Language-Programmquellcodedatei.</td></tr>
</table>

ul.ulh	(Hypertextlink zu) **Bartels User Language**-Includedatei.
ULPROG	(Hypertextlink zu) **Bartels User Language**-Programmbeschreibung.
ul_function	(Hypertextlink zu) **Bartels User Language**-Systemfunktionsbeschreibung.
UL_INDEX	(Hypertextlink zu) **Bartels User Language**-Indextypbeschreibung.
UTILPROG	(Hypertextlink zu) **Bartels AutoEngineer** Utilityprogrammbeschreibung.
neue Funktion	Neue Funktionen die mit regulären wöchentlichen Updates/Builds verfügbar gemacht werden, werden in der Onlinedokumentation hervorgehoben.

In Beispielen für höhere Programmiersprachen und Interpretationssprachen, in Objektbeschreibungen mittels Spezifikationssprachen, bei der Verwendung in Syntaxbeschreibungssprachen, etc. erhalten obige Sonderzeichensequenzen wieder die in der entsprechenden Sprache festgelegte Bedeutung.

Inhalt

Tabellen

Kapitel 1 Einleitung

Dieses Kapitel enthält einleitende Anmerkungen zur **Bartels User Language**. Hierbei werden der Verwendungszweck der **Bartels User Language** erläutert sowie deren Bestandteile vorgestellt. Außerdem wird auf die grundlegenden Eigenschaften dieser Programmiersprache eingegangen.

Inhalt

1.1 Was ist Bartels User Language?

1.1.1 Verwendungszweck

Durch den Einsatz der **Bartels User Language** ergeben sich praktisch unbegrenzte Möglichkeiten für den Zugriff auf die Datenbankinhalte sowie die Nutzung von Funktionen des **Bartels AutoEngineer**. Mit Hilfe der **Bartels User Language** lassen sich im **Bartels AutoEngineer** unter anderem

- CAM-Postprozessoren zur Ausgabe beliebiger Daten(formate) erzeugen
- anwenderspezifische Menüfunktionen ("Makros") implementieren und integrieren
- spezielle Reportfunktionen bereitstellen
- Funktionen zur Prüfung definierbarer Designregeln einführen
- automatische Routinen zur Bibliotheksbearbeitung entwickeln
- Automatismen zur Bauteilplatzierung oder für das Routing implementieren
- Programme für den CAM-Batchbetrieb bereitstellen
- Anwendungen des **Neuronale Regelsystems** implementieren
- firmenspezifische, relationale Datenbanken einbinden
- Werkzeuge zur Übernahme bzw. Ausgabe von Fremddaten implementieren

User Language-Programme können transparent in das BAE-Menüsystem eingebunden werden, und die Konfiguration der BAE-Software zum Aufruf häufig benötigter **User Language**-Programme über Tastendruck (Hotkey) ist ebenfalls möglich.

Da nicht jeder BAE-Anwender notgedrungen ein erfahrener Softwareentwickler sein muss, erwarten wir von unseren Kunden auch nicht, dass sie selbst extensive **User Language**-Programmierung betreiben (wenngleich sie dies natürlich tun können). Vielmehr versetzt *uns* das Konzept der **User Language** in die Lage, in kürzester Zeit und mit einem Höchstmaß an Flexibilität praktisch beliebige Anpassungen der BAE-Software an kundenspezifische Bedürfnisse vorzunehmen, ohne dass dadurch Änderungen im BAE-Software-Kernel und damit ein organisatorisch aufwändiger Software-Update notwendig wären. Damit können wir unseren Kunden die bestmögliche Unterstützung im Hinblick auf die Implementierung gewünschter Spezial- bzw. Zusatzfunktionen bieten. Sichtbares Resultat dieses einzigartigen Konzepts sind die zahlreichen, nach Kundenwünschen implementierten **User Language**-Programme, die integraler Bestandteil der BAE-Software sind. Da die **User Language**-Programme der BAE-Software im Quellcode ausgeliefert werden, kann der BAE-Anwender Anpassungen an firmenspezifische Bedürfnisse leicht selbst vornehmen. In diesem Zusammenhang sei auf das Kapitel 4 dieses Handbuchs verwiesen, welches Kurzbeschreibungen zu den mit der BAE-Software ausgelieferten **User Language**-Programmen sowie Informationen zur Installation bzw. Bereitstellung dieser Programme im **Bartels AutoEngineer** enthält.

1.1.2 Bestandteile

Die **Bartels User Language** besteht aus der Definition der Programmiersprache selbst, dem **User Language Compiler**, sowie dem **User Language Interpreter**.

Definition der User Language Programmiersprache

Bartels User Language ist eine auf dem Sprachumfang von C basierende Programmiersprache mit internen, aus der objektorientierte Programmierung (OOP) bekannten Erweiterungen (automatische Speicherverwaltung bei der Bearbeitung von Listen, spezieller Datentyp für Zeichenketten). Über spezielle Variablentypen ermöglicht **Bartels User Language** den Zugriff auf die Design-Datenbank (DDB) des **Bartels AutoEngineer**. Eine eingebundene Funktionsbibliothek erlaubt den Aufruf von Standard- und BAE-Systemfunktionen. Die **User Language** Programmiersprache ist in Kapitel 2 ausführlich beschrieben. Die Definition aller Variablentypen für den DDB-Zugriff ist in Anhang B dokumentiert. Anhang C enthält die vollständige Beschreibung der in die **User Language** eingebundenen Systemfunktionsbibliothek.

Bartels User Language Compiler

Der **Bartels User Language Compiler (ULC)** erlaubt die Übersetzung von **Bartels User Language**-Quelldateien in den für den **Bartels User Language Interpreter** ausführbaren Code. Der Compiler führt bei der Übersetzung eine ganze Reihe von Prüfungen hinsichtlich Datentypkompatibilität und Ausführbarkeit von Programmen durch und arbeitet wahlweise optimierend. Spezielle Compiler-Optionen erlauben die wahlweise Generierung von **User Language**-Libraries. Der integrierte Linker des **User Language Compilers** ermöglicht sowohl das statische Linken von **User Language**-Libraries (zur Compile-Zeit) als auch die Programmvorbereitung zum dynamischen Linken (während der Laufzeit). Eine ausführliche Beschreibung des **User Language Compilers** ist in Kapitel 3.2 enthalten.

Bartels User Language Interpreter

Der **Bartels User Language Interpreter** erlaubt (das dynamische Linken und) die Ausführung von kompilierten, d.h. durch den **Bartels User Language Compiler** generierten **User Language**-Programmen. Der **Bartels User Language Interpreter** ist derzeit in den **Schaltplaneditor**, den **Layouteditor**, den **Autorouter**, den **CAM-Prozessor**, das **CAM-View**-Modul und den **Chipeditor** des **Bartels AutoEngineer** eingebunden. D.h., aus diesen Programmteilen des **Bartels AutoEngineer** heraus ist der Start von **User Language**-Programmen möglich. Der Programmaufruf kann dabei wahlweise über eine spezielle Menüfunktion durch explizite Angabe des Programmnamens oder automatisiert über Tastendruck bzw. Programm-Modul-Startup erfolgen. Eine ausführlichere Beschreibung des **User Language Interpreter** finden Sie im Kapitel 3.3 dieses Handbuchs.

1.2 Charakteristische Eigenschaften der User Language

1.2.1 Bartels User Language im Vergleich zu C

Wie bereits einleitend erwähnt, ist **Bartels User Language** eine Programmiersprache, die auf dem Sprachumfang von C basiert. Kommentare sind wie in C üblich durch `/*` und `*/` zu begrenzen bzw. werden wie in C++ üblich mit `//` eingeleitet und durch das Zeilenende begrenzt.

Als elementare Datentypen sind `char`, `int` und `double` in **Bartels User Language** enthalten. Im Unterschied zu C fehlt hier der Datentyp `float` ebenso wie die Möglichkeit der Qualifizierung elementarer Datentypen (`short`, `unsigned`, `long`). Zeiger sind in der **Bartels User Language** nicht implementiert. An Stelle von `char`-Vektoren zur Darstellung von Zeichenketten kann in **Bartels User Language** der als elementar zu betrachtende Datentyp `string` verwendet werden. In Erweiterung zu C enthält **Bartels User Language** den ebenfalls als elementar zu betrachtenden speziellen Datentyp `index`. Über diesen Datentyp werden die Möglichkeiten des Zugriffs auf die Design-Datenbank (DDB) des **Bartels AutoEngineer** definiert. `index` kann dabei als Index in einen Vektor von DDB-Strukturen verstanden werden. Über einen speziellen Operator ist der Zugriff auf die Elemente der darüber adressierten Struktur möglich. Die `index`-Typen sowie die zugehörigen Strukturelemente sind vordefiniert (siehe Anhang B).

Vektoren werden im Gegensatz zu C dynamisch verwaltet, d.h. die Definition fester Vektorfeldgrenzen entfällt in der **User Language**. Ebenso besteht in der **User Language** die Möglichkeit, zusammengesetzte, komplexe Datentypen (Strukturen, Arrays) einander direkt zuzuweisen, sofern diese typkompatibel sind und dieselbe Dimensionierung aufweisen.

Die Vereinbarung der aus C bekannten Speicherklassen `auto`, `extern` und `register` ist in der **Bartels User Language** nicht explizit möglich. Grundsätzlich werden in der **User Language** alle innerhalb von Funktionen definierten Variablen der Speicherklasse `auto` zugeordnet, alle außerhalb von Funktionen definierten Variablen gelten als globale Variablen, sofern sie nicht explizit der Speicherklasse `static` zugeordnet sind. Ebenso gelten alle im Programmtext definierten Funktionen als global vereinbart, sofern sie nicht explizit der Speicherklasse `static` zugeordnet sind. Der Geltungsbereich globaler Variablen und Funktionen erstreckt sich auf das gesamte Programm, während als `static` deklarierte Variablen und Funktionen nur Gültigkeit im aktuellen Programmtext (nicht jedoch in einem noch zu linkenden Programmtext) besitzen.

Mechanismen zur Spracherweiterung mit Hilfe eines Makro-Preprozessors werden in **Bartels User Language** ebenfalls unterstützt; so stehen die aus C bekannten Preprozessor-Anweisungen `#include`, `#define`, `#undef`, `#if`, `#ifdef`, `#ifndef`, `#else` und `#endif` auch in der **User Language** zur Verfügung.

1.2.2 Datentypen, Konstanten, Variablen

Bartels User Language enthält die *elementaren Datentypen* `char` (Zeichen), `int` (ganzzahliger Wert), `double` (rationale Zahl doppelter Genauigkeit), `string` (Zeichenkette), sowie `index` (vordefinierter Index auf DDB-Struktur; siehe Anhang B). Daneben lassen sich aus den elementaren Datentypen abgeleitete Datentypen (Vektoren bzw. Arrays sowie Strukturen) definieren.

Bartels User Language erlaubt die Verwendung von *Konstanten* für die elementaren Datentypen. `char`-Konstanten sind dabei durch einfache, `string`-Konstanten durch doppelte Anführungszeichen zu begrenzen. Die in C übliche Verwendung des Backslashs (`\`) als Fluchtsymbol zur Angabe grafisch nicht darstellbarer Zeichen ist zulässig. Ganzzahlige Konstanten können wahlweise in Dezimal-, Oktal- oder Hexadezimalschreibweise dargestellt werden. Festkommakonstanten müssen einen Dezimalpunkt enthalten. Daneben ist auch die wissenschaftliche Gleitkommadarstellung mit Exponent erlaubt.

Konstante Ausdrücke bestehen aus Konstanten und Operatoren und werden bereits während der Übersetzung durch den **Bartels User Language Compiler** bewertet (Constant Expression Evaluation).

Alle *Variablen* müssen vereinbart werden, wodurch jeweils der Datentyp der Variablen, sowie deren Name festgelegt wird. Variablennamen müssen mit einem Buchstaben oder einem Unterstrich (_) beginnen und können nachfolgend beliebig viele Buchstaben, Ziffern oder Unterstriche aufweisen. Der Compiler unterscheidet bei der Verwendung von Buchstaben in Variablennamen zwischen Groß- und Kleinschreibung. Variablen können bereits bei deren Vereinbarung initialisiert werden. Wird eine nicht initialisierte Variable verwendet, dann gibt der Compiler ggf. eine entsprechende Warnmeldung aus. Der Interpreter hingegen wird in solchen Fällen die entsprechende Variable mit einem Nullwert initialisieren.

1.2.3 Operatoren, Zuweisungen

Bartels User Language verfügt über die aus C bekannten Operatoren (`?:`, `+`, `-`, `*`, `/`, `%`, `>`, `>=`, `<`, `<=`, `==`, `!=`, `&&`, `||`, `!`, `++`, `--`, `&`, `|`, `^`, `<<`, `>>`, `~`). Die Reihenfolge des Vorrangs für die Bewertung von Operatoren entspricht ebenfalls der Programmiersprache C. Erscheinen Operanden mit verschiedenen Datentypen zusammen in Ausdrücken, dann werden deren Werte in einen gemeinsamen Datentyp umgewandelt. Derartige Umwandlungen finden allerdings nur dann statt, wenn sie möglich und auch sinnvoll sind (ansonsten Fehlermeldung durch den Compiler). Eine Erweiterung gegenüber C ergibt sich aus der Möglichkeit, den Additionsoperator `+` sowie die Vergleichsoperatoren (`>`, `>=`, `<`, `<=`, `==`, `!=`) direkt auf den Datentyp `string` anzuwenden.

Eine Zuweisung erfolgt üblicherweise durch den `=`-Operator, wobei der Ausdruck auf der rechten Seite des Gleichheitszeichens der Variablen auf der linken zugewiesen wird. Die in C üblichen zusammengesetzten Zuweisungsoperatoren (`+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`, `<<=`, `>>=`) sind auch in **Bartels User Language** zulässig.

1.2.4 Kontrollstrukturen

Durch Kontrollstrukturen wird die Reihenfolge der auszuführenden Aktionen definiert. **Bartels User Language** erlaubt alle aus C bekannten Kontrollstrukturen außer der `goto`-Anweisung sowie der Definition von Marken. Die verfügbaren Kontrollstrukturen sind somit `if` bzw. `if-else`, `switch`, `while`, `for`, `do-while`, `break` und `continue`. Eine Erweiterung gegenüber C bildet die `forall`-Kontrollstruktur, mit der ein repetitives Abarbeiten von `index`-Datentyp-Variablen möglich ist.

1.2.5 Programmfluss, Funktionen

Wie C ermöglicht auch **Bartels User Language** die Zerlegung großer Problemstellungen in kleinere mit Hilfe von Funktionen. Zu unterscheiden ist dabei zwischen den in **Bartels User Language** eingebundenen Systemfunktionen (siehe Anhang C) und den Funktionen, die der Programmierer innerhalb eines Programms selbst definiert (Anwenderfunktionen). Die Definition und Deklaration der Anwenderfunktionen mit ihren Funktionsparametern entspricht der Vorgehensweise in C. Ebenso ist die Anwenderfunktion mit dem Namen `main` immer die erste Funktion, die bei einem Programmlauf aufgerufen wird (d.h. die Funktion `main` muss i.d.R. definiert sein, damit ein **User Language**-Programm überhaupt "etwas tut"). Im Gegensatz zu C unterscheidet **Bartels User Language** bei der Bewertung von an Funktionen übergebenen Parametern nicht zwischen "Call-By-Value" und "Call-By-Reference"; alle Parameter werden jeweils durch das "Call-By-Reference"-Verfahren bewertet, wodurch selbstverständlich nicht (wie sonst in C üblich) mit Zeigern gearbeitet werden muss, um geänderte Parameterwerte an einen Funktionsaufrufer zurückzumelden.

1.2.6 Integrierte Spezialfunktionen

Bartels User Language stellt dem Anwender einige spezielle Werkzeuge zur Verfügung, die es aufgrund ihres umfangreichen Funktionsspektrums und ihrer außerordentlichen Mächtigkeit verdienen, an dieser Stelle gesondert erwähnt zu werden.

So ist in die **Bartels User Language** ein BNF-Precompiler zur Realisierung von Interfaceprogrammen für die Bearbeitung von Fremddatenformaten integriert. Unter Verwendung der zugehörigen Scanner- und Parserfunktionen lassen sich in einfacher und eleganter Weise Programme zur Verarbeitung praktisch beliebiger ASCII-Datenformate implementieren (siehe hierzu auch Kapitel 2.6.4 dieses Handbuchs).

Bartels User Language enthält des Weiteren SQL (Structured Query Language)-Funktionen zur Verwaltung Relationaler Datenbanken. Damit stehen dem Anwender Softwaretools zur Programmierung von Datenbankmanagementsystemen zur Verfügung. Die entsprechenden SQL-Zugriffsfunktionen erlauben es, z.B. eine Datenbank für Bauteildaten zur Variantenverwaltung in den **Bartels AutoEngineer** integrieren. Damit lassen sich in einfacher Weise "was wäre wenn"-Analysen (Kosten, Lagerbestand, usw.) für verschiedene Varianten eines Layouts durchführen, sowie komfortable Datenbankfunktionen zur schnellen Auswahl geeigneter Bauteile mit einer kontrollierten Zuordnung von Gehäusebauform und Bauteilwert implementieren. Dies ist jedoch nur ein Beispiel aus dem breiten Anwendungsspektrum; selbstverständlich lassen sich auch Datenbanksysteme zur Projektverwaltung, zum Projektmanagement, zur Versionsverwaltung, zur Produktionsplanung und -steuerung (PPS), zur Adressverwaltung, zur Verwaltung von Lieferantenverzeichnissen und Kundendateien, usw. realisieren. Anhang C dieses Handbuchs enthält die Beschreibungen der SQL-Systemfunktionen.

Kapitel 2
Sprachbeschreibung

Dieses Kapitel enthält die Sprachbeschreibung der **Bartels User Language**. Hierbei wird detailliert auf die Elemente der **Bartels User Language** eingegangen, und wo nötig wird deren Verwendung durch Beispiele veranschaulicht. Das Kapitel enthält außerdem Erläuterungen zur Bedienung der Programmierumgebung sowie zur Schnittstelle zum **Bartels AutoEngineer**.

Inhalt

Tabellen

2.1 Einführung in die User Language Programmierung

An dieser Stelle sollen anhand kleiner Programmbeispiele die wichtigsten Sprachelemente der **User Language** kurz vorgestellt werden, ohne zunächst Wert auf Vollständigkeit zu legen bzw. auf Details oder gar Ausnahmeregelungen einzugehen. Ziel dabei ist, möglichst schnell die Vorgehensweise bei der **User Language** Programmierung aufzuzeigen.

2.1.1 Ein erstes User Language Programm

Als erstes soll ein Programm erstellt werden, welches lediglich eine Meldung ausgibt und dann auf eine Eingabe wartet, um den Programmlauf abzubrechen. Dies ist im Übrigen bereits ein Programmkonstrukt, das innerhalb des **Bartels AutoEngineer** relativ häufig benötigt wird. Wie Sie bereits aus der Einleitung wissen, muss ein **User Language**-Programm mindestens aus der `main`-Funktion bestehen. Was innerhalb der `main`-Funktion benötigt wird, ist eine Anweisung, die die gewünschte Meldung ausgibt, sowie eine Anweisung, die eine Benutzerabfrage aktiviert. Beide Anweisungen werden durch den Aufruf entsprechender **User Language** Systemfunktionen (**printf** und **askstr**) realisiert. Diese Systemfunktionen sind dem Compiler bekannt, und in den Interpreter eingebunden, d.h. der Programmierer muss lediglich wissen, wie diese Funktionen aufzurufen sind, und was sie tun (diese Information kann dem Anhang C dieses Handbuchs entnommen werden). Erstellen Sie nun mit Ihrem Texteditor folgendes **User Language**-Programm und speichern Sie dieses unter dem Dateinamen **`ulprog.ulc`** ab (an der File-Extension **`.ulc`** erkennt der Compiler, dass es sich um ein User Language Programm handelt):

```
main()
{
      printf("User Language Program");
      askstr("Press ENTER to continue ",1);
}
```

Wie Sie sehen, besteht das Programm lediglich aus der Definition der Funktion **`main`**. Innerhalb der runden Klammern nach dem Funktionsnamen stehen normalerweise die formalen Parameter der Funktion. Um den Funktionsnamen von anderen Variablennamen zu unterscheiden, sind diese Klammern auch anzugeben, wenn, wie in obigem Beispiel, keine Parameter existieren. Innerhalb der geschweiften Klammern befindet sich der Rumpf der Funktion, d.h. die Anweisungen, die die Funktion ausführen soll. Jede Anweisung wird durch ein Semikolon (**`;`**) abgeschlossen. Die erste Anweisung innerhalb der Funktion **`main`** ist der Aufruf der Funktion **printf**, was an der nachfolgenden runden Klammer zu erkennen ist. Als Argument bzw. Parameter wird an **printf** (in doppelten Anführungszeichen) eine konstante Zeichenkette übergeben, die das Programm bei fehlerfreier Übersetzung und Ausführung am Bildschirm ausgibt. Die zweite Anweisung ist ein Aufruf der Funktion **askstr**. Diese Funktion gibt in der Eingabe- bzw. Mitteilungszeile des **Bartels AutoEngineer** ihr erstes Argument in Form eines Prompts aus und wartet auf die Eingabe einer Zeichenkette durch den Anwender. Der zweite Parameter zu **askstr** gibt dabei die maximal zulässige Länge der einzugebenden Zeichenkette an. Der Aufruf von **askstr** ist zugleich die letzte Anweisung des Programms, d.h. nach der Bearbeitung des **askstr**-Aufrufs wird das Programm beendet. Wenn das Programm fertig codiert und unter **`ulprog.ulc`** abgespeichert ist, kann es mit folgendem Aufruf des **User Language Compilers** übersetzt werden:

```
ulc ulprog
```

Falls keine Fehler auftreten, gibt der Compiler die folgende Meldung auf dem Bildschirm aus:

```
==============================
BARTELS USER LANGUAGE COMPILER
==============================

Quellcodedatei "ulprog.ulc" wird kompiliert...
Programm 'ulprog' erfolgreich generiert.
Quellcodedatei "ulprog.ulc" erfolgreich kompiliert.

Keine Fehler, keine Warnungen.
User Language Compiler-Lauf erfolgreich beendet.
```

Der Compiler hat das Programm übersetzt und unter dem Namen `ulprog` in der Datei `ulcprog.vdb` im **Bartels AutoEngineer** Programmverzeichnis abgespeichert. Jetzt kann das Programm durch den **User Language Interpreter** ausgeführt werden. Hierzu ist z.B. der **Schaltplaneditor** des **Bartels AutoEngineer** aufzurufen und die Funktion Anwenderfunktion im Menü Datei zu aktivieren. Auf die Abfrage nach dem Namen des auszuführenden Programms ist anschließend `ulprog` einzugeben:



Der Grafikarbeitsbereich des **AutoEngineer** wird in den Textmodus geschaltet und es wird die Meldung `User Language Program` angezeigt. Anschließend wird im Eingabefenster der Prompt `Press ENTER to continue` angezeigt. Betätigt der Anwender daraufhin die Return-Taste, dann wird das **User Language**-Programm beendet und der Grafikarbeitsbereich wieder in den Grafikmodus zurückgeschaltet.

2.1.2 Variablen, Arithmetik und Funktionen

Anhand des nächsten Beispiels sollen eine ganze Reihe weiterer spezifischer Eigenschaften der **User Language** veranschaulicht werden. Das folgende Programm überprüft einige durch ihren Mittelpunkt und ihren Radius definierte Kreise daraufhin, ob sie sich überlappen (Bohrdatentest?!), und gibt entsprechende Meldungen aus:

```
// Circle Test Program

double tol=0.254*5;           // Tolerance

struct pos {                  // Position descriptor
      double x;               // X coordinate
      double y;               // Y coordinate
      };

struct circle {               // Circle descriptor
      double rad;             // Circle radius
      struct pos c;           // Circle position
      };

// Main program
main()
{
      // Define three circles
      struct circle c1 = {  4.5, { 19.4, 28.3} };
      struct circle c2 = { 17.0, { 37.6, 9.71} };
      struct circle c3 = { 1.5E01, { 25, 0.2e2} };
      // Perform circle test
      printf("Circle 1 - 2 overlap : %d\n",circletest(c1,c2));
      printf("Circle 1 - 3 overlap : %d\n",circletest(c1,c3));
      printf("Circle 2 - 3 overlap : %d\n",circletest(c2,c3));
      // Prompt for continue
      askstr("Press ENTER to continue ",1);
}

int circletest(c1,c2)
// Circle test function
// Returns: nonzero if overlapping or zero else
struct circle c1,c2           /* Test circles 1 and 2 */;
{
      double d                /* Distance value */;
      // Get circle center point distances
      d=distance(c1.c,c2.c);
      // Error tolerant check distance against radius sum
      return(d<=(c1.rad+c2.rad+tol));
}

double distance(p1,p2)
// Get distance between two points
// Returns: distance length value
struct pos p1                 /* Point 1 */;
struct pos p2                 /* Point 2 */;
{
      double xd=p2.x-p1.x     /* X distance */;
      double yd=p2.y-p1.y     /* Y distance */;
      // Calculate and return distance
      return(sqrt(xd*xd+yd*yd));
}
```

Obiger Quelltext enthält eine Reihe von Kommentaren, die durch `/*` und `*/` eingeklammert sind; diese Kommentare dürfen sich über mehrere Zeilen erstrecken, aber sie dürfen nicht verschachtelt werden. Ein anderer Typ von Kommentar beginnt mit `//` und erstreckt sich jeweils bis zum Zeilenende. Sie sollten sich angewöhnen, Ihre Programme mit derartiger Inline-Dokumentation zu versehen, damit der Programmcode gut verständlich bleibt und somit - auch durch dritte - leichter gepflegt bzw. weiterentwickelt werden kann.

Das Programm enthält weiterhin eine Reihe von Variablendefinitionen. Variablen müssen grundsätzlich vor ihrer Verwendung definiert werden. Dabei wird sowohl der Datentyp als auch der Variablenname festgelegt. Unterschieden wird zwischen globalen Variablen, lokalen Variablen und Funktionsparametern. Der Geltungsbereich globaler Variablen erstreckt sich über das gesamte Programm. Lokale Variablen sind nur innerhalb der Funktion, in der sie definiert wurden, gültig. Funktionsparameter dienen dazu, Werte an Funktionen zu übergeben. In obigem Beispiel wird als einzige globale Variable **`tol`** mit dem Datentyp **`double`** definiert. Beispiele für lokale Variablen sind die **`double`**-Variablen **`xd`** und **`yd`** in der Funktion **`distance`**. Funktionsparameter sind z.B. **`c1`** und **`c2`** in der Funktion **`circletest`**. Diese sind von dem speziell definierten zusammengesetzten **`struct`**-Datentyp **`circle`**. Initialisierungen globaler und lokaler Variablen lassen sich bereits bei deren Deklaration durchführen. Beispiele hierfür sind die globale Variable **`tol`** sowie die lokalen Variablen **`xd`** und **`yd`** der Funktion **`distance`**. Auch zusammengesetzte Datentypen lassen sich initialisieren, wie die Beispiele für die lokalen **`struct`**-Variablen **`c1`**, **`c2`** und **`c3`** der Funktion **`main`** zeigen. Bei der Variablendeklaration besteht darüber hinaus die Möglichkeit, gleich eine ganze Liste von Variablennamen anzugeben (siehe Parameterdeklaration für **`c1`** und **`c2`** in der Funktion **`circletest`**).

Die Berechnung von Werten erfolgt über Ausdrücke, wobei das Gleichheitszeichen (**`=`**) als Zuweisungsoperator fungiert.

Bei der Definition von Funktionen ist ebenfalls ein Datentyp zu spezifizieren. In obigem Beispiel ist die Funktion **`distance`** vom Typ **`double`**, die Funktion **`circletest`** vom Typ **`int`**. Wird die Datentypspezifikation - wie bei der Funktion **`main`** - weggelassen, dann wird deren Typ automatisch auf **`int`** gesetzt. Ein spezieller Datentyp für Funktionen ist **`void`**. Jede Funktion außer den **`void`**-Funktionen liefert einen zum Funktionsdatentyp kompatiblen Rückgabewert. Die Übergabe des Rückgabewerts geschieht mit der **`return`**-Funktion, welche gleichzeitig die Funktionsausführung beendet.

2.1.3 Vektoren und Kontrollstrukturen

An dieser Stelle sei abschließend ein Beispiel für die Verwendung von Vektoren und Kontrollstrukturen aufgeführt. Das nachfolgende Programm wandelt eine ganze Liste von Integerwerten in Strings um und gibt diese aus:

```
// Integer list
int intary[]={ 0,17,-12013,629,0770,0xFF,-16*4+12 };

// Main program
main()
{
      int i                   /* Loop control variable */;
      // Set last integer value
      intary[10]=(-1);
      // Loop through integer list
      for (i=0;i<=10;i++)
              // Print integer and integer string
              printf("%8d : \"%s\"\n",intary[i],inttostr(intary[i]));
      // Prompt for continue
      askstr("Press ENTER to continue ",1);
}

string inttostr(int intval)
// Convert integer value to a string
// Returns: resulting string
{
      string resstr=""        /* Result string */;
      int n=intval,i=0        /* Integer value, loop counter */;
      char sign               /* Sign character */;
      // Test for negative integer value
      if (n==0)
              // Return zero integer string
              return("0");
      else if (n>0)
              // Set sign to plus character
              sign='+';
      else {
              // Make integer value positive
              n=-n;
              // Set sign to minus character
              sign='-';
              }
      // Build result string
      do {    // Get and append next character
              resstr[i++]=n%10+'0';
              } while ((n/=10)!=0);
      // Append zeros
      while (i++<15)
              resstr+='0';
      // Append sign character
      resstr+=sign;
      // Reverse string
      strreverse(resstr);
      // Return string result
      return(resstr);
}
```

In obigem Programm wird ein Vektor aus Integerwerten (globale `int`-Variable `intary`) deklariert und dabei auch bereits (teilweise) initialisiert. Wie leicht zu erkennen ist, definiert das eckige Klammernpaar nach dem Variablennamen `intary` einen eindimensionalen `int`-Vektor. Mehrdimensionale Vektoren sind durch entsprechendes Anfügen weiterer eckiger Klammernpaare (`intary[][]...[]`) zu deklarieren. Zu beachten ist dabei, dass die Angabe einer Feldlänge entfällt. Dies resultiert aus der Fähigkeit der **User Language**, Vektoren dynamisch zu verwalten, d.h. sowohl dem Compiler als auch dem Interpreter genügt die Information über die Dimension des Vektors. Allerdings erfolgen doch einige Prüfungen, die den Zugriff auf nicht existente Vektorelemente (und damit eine Speicherzugriffsverletzung) unterbinden. So erkennt der Compiler z.B., wenn versucht wird, über einen konstanten, negativen (also ungültigen) Index auf ein Vektorelement zuzugreifen. Ebenso prüft der Interpreter Vektorindizes, die außerhalb des aktuell belegten Vektorfeldbereiches liegen. Beachten Sie hierbei, dass der Index mit dem Wert 0 auf das erste Element eines Vektors verweist.

Eine spezielle Form eines Vektors stellt der Datentyp **`string`** dar. **User Language** erlaubt die direkte Zuweisung von Vektoren kompatiblen Datentyps und gleicher Dimension. Dies wurde bei der Initialisierung der lokalen Variable **`resstr`** in der Funktion **`inttostr`** ebenso wie bei der Definition dieser Funktion als **`string`** und beim Setzen entsprechender Rückgabewerte in den **`return`**-Aufrufen ausgenutzt. Zudem lässt sich der Additionsoperator direkt auf **`string`**-Datentypen anwenden; das Resultat entspricht dabei einem durch Aneinanderfügen der beiden Additionsoperatoren erzeugten **`string`**.

Obiges Programmbeispiel enthält einige Kontrollstrukturen. So wird in der Funktion **`main`** über eine **`for`**-Schleife die Vektorvariable **`intary`** abgearbeitet. In der Funktion **`inttostr`** werden eine **`while`**- und eine **`do-while`**-Schleife verwendet, um die **`string`**-Variable **`resstr`** zu manipulieren. Des Weiteren enthält die Funktion **`inttostr`** eine **`if`**-Kontrollstruktur, die in Abhängigkeit vom Wert der lokalen Variablen **`n`** jeweils in einen entsprechenden Programmblock verzweigt.

2.2 Konventionen

In der **User Language** sind die Wortklassen Zwischenraum, Identifier, Konstante, reserviertes Wort und Operator definiert.

2.2.1 Zwischenraum

Unter die Wortklasse Zwischenraum fallen sowohl Leerstellen, Tabulatorzeichen, Zeilentrenner als auch Kommentare. Kommentare beginnen mit den Zeichen `/*` und enden mit `*/`; diese Kommentare dürfen nicht verschachtelt sein. Ein weiterer Typ von Kommentar beginnt mit den Zeichen `//` und erstreckt sich bis zum Zeilenende. Der **User Language Compiler** wertet Zwischenräume lediglich zur Trennung direkt benachbarter Identifier und reservierter Worte aus; ansonsten werden alle Zwischenräume ignoriert.

2.2.2 Identifier

Ein Identifier ist der Name einer Variablen, einer Funktion oder einer symbolischen Konstanten. Jeder Identifier darf aus einer Folge von Buchstaben und Ziffern bestehen, wobei das erste Zeichen immer ein Buchstabe sein muss. Der Unterstrich (`_`) zählt dabei zu den Buchstaben. Der **User Language Compiler** unterscheidet bei der Auswertung von Identifiern zwischen Groß- und Kleinbuchstaben. Es besteht die Einschränkung, dass Identifier nicht mit reservierten Worten (siehe unten) identisch sein dürfen.

Beispiele:

```
X_coord   value   P4   File_Name   _euklid
```

2.2.3 Konstanten und konstante Ausdrücke

Nachfolgend sind die in der **User Language** definierten Konstanten beschrieben. Jeder Konstanten-Typ ist zugleich auch einem Datentyp zugeordnet.

Ganzzahlige Konstanten

Ganzzahlige Konstanten sind dem Datentyp `int` zugeordnet. Sie bestehen aus einer Folge von Ziffern und werden üblicherweise dezimal interpretiert. Falls die Ziffernfolge mit `0` (Ziffer Null) beginnt, wird sie oktal, d.h. in Basis 8 interpretiert (die Ziffern 8 und 9 sind in diesem Fall nicht zulässig). Beginnt die Ziffernfolge mit `0x` bzw. `0X` (Ziffer Null, Buchstabe x), dann wird sie hexadezimal, d.h. in Basis 16 interpretiert, wobei dann die Buchstaben `a` bis `f` bzw. `A` bis `F` als hexadezimale Ziffern mit den Werten 10 bis 15 gelten. Ganzzahlige negative Konstanten gibt es nicht; vielmehr stellt das Minus-Zeichen einen Operator dar, der in Verbindung mit einer ganzzahligen Konstante einen konstanten Ausdruck (siehe unten) bildet.

Beispiele:

```
1432   073   0xF4A5   9
```

Gleitkomma-Konstanten

Gleitkomma-Konstanten sind dem Datentyp `double` zugeordnet. Sie bestehen aus einem ganzzahligen Teil, einem Dezimalpunkt, einem Dezimalbruch, dem Zeichen `e` bzw. `E` (Buchstabe `e`) und einem ganzzahligen Exponenten mit optionalem Vorzeichen. Ganzzahliger Teil, Dezimalbruch sowie Exponent sind dabei Ziffernfolgen. Genau einer der Teile vor oder nach dem Dezimalpunkt (`.`) darf fehlen; entweder der Dezimalpunkt oder der Exponent beginnend mit dem Buchstaben e darf fehlen.

Beispiele:

```
2.54   .78   4.   4.1508E-3   0.81037e6   17228E5
```

Zeichen-Konstanten

Zeichen-Konstanten sind dem Datentyp `char` zugeordnet. Sie bestehen aus einem einzelnen, in einfache Anführungsstriche (Apostroph) eingeschlossenen Zeichen. Der Wert dieser Konstante ist der numerische Wert des Zeichens im Zeichensatz der Maschine, auf der das **User Language**-Programm ausgeführt wird.

Tabelle 2-1 enthält eine Liste von Sonderzeichen-Darstellungen mit Hilfe des Fluchtsymbols `\` (Backslash).

Tabelle 2-1: Darstellung von Sonderzeichen

Rückwärtsschritt	BS	`\b`
Horizontal Tabulator	HT	`\t`
Zeilenvorschub	LF	`\n`
Seitenvorschub	FF	`\f`
Wagenrücklauf	CR	`\r`
Fluchtsymbol	\	`\\`
Apostroph	'	`\'`
Nullzeichen	NUL	`\0`

Beliebige Bitmuster können durch die Angabe des Fluchtsymbols, gefolgt von bis zu drei oktalen Ziffern spezifiziert werden. Ein Spezialfall dieser Konstruktion ist das Nullzeichen (NUL, `\0`).

Zeichenketten-Konstanten

Zeichenketten-Konstanten (oder auch String-Konstanten) sind dem Datentyp **`string`** zugeordnet. Sie bestehen aus einer Folge von Zeichen umgeben von Doppel-Anführungsstrichen. Der Compiler legt ein NUL Zeichen (**`\0`**) am Ende der String-Konstanten ab, damit der Interpreter das Ende des konstanten Strings finden kann. Bei der Spezifikation von String-Konstanten muss auch der Doppel-Anführungsstrich mit einem Fluchtsymbol angegeben werden. Daneben existieren dieselben Fluchtsymbol Kombinationen wie für die Zeichen-Konstanten.

Beispiele:

```
"IC1"   "4.8 kOhm"   "This is a string with Newline\n"
```

Konstante Ausdrücke

Ein konstanter Ausdruck ist ein Ausdruck, der nur aus konstanten Werten und Operatoren zusammengesetzt ist. Solche Ausdrücke werden bereits vom Compiler bewertet (CEE, Constant Expression Evaluation) und müssen somit nicht erst zur Laufzeit durch den Interpreter berechnet werden. Dies bedeutet, dass, wo immer Konstanten benötigt werden, statt dessen auch entsprechende konstante Ausdrücke verwendet werden können, ohne dass sich dadurch Nachteile hinsichtlich der Programmgröße oder der Programmlaufzeit ergeben.

Beispiele:

```
int i=19-(010+0x10);                CEE: int i=-5;
double d=-(4.7+2*16.3);             CEE: double d=-37.3;
string s="Part"+' '+"IC1";          CEE: string s="Part IC1";
```

2.2.4 Terminalzeichen-Sequenzen

Reservierte Worte

Tabelle 2-2 enthält die Liste der Identifier, die in der **Bartels User Language** als Schlüsselwörter reserviert sind. Diese Identifier dürfen nur in ihrer vordefinierten Bedeutung verwendet werden:

Tabelle 2-2: Reservierte Worte

#bnf	#define	#else	#endif	#if	#ifdef	#ifndef	#include
#undef	break	case	char	continue	default	do	double
else	for	forall	if	index	int	of	return
static	string	struct	switch	typedef	void	where	while

Operatoren

Die in Tabelle 2-3 aufgeführten Zeichen(-Sequenzen) sind in der **Bartels User Language** als Operatoren definiert und lösen je nach Kontext spezielle Aktionen aus:

Tabelle 2-3: Operatoren

<table>
<tr><td>!</td><td>!=</td><td>%</td><td>%=</td><td>&</td><td>&&</td><td>&=</td><td>(</td><td>)</td><td>*</td><td>*=</td></tr>
<tr><td>+</td><td>++</td><td>+=</td><td>,</td><td>-</td><td>--</td><td>-=</td><td>.</td><td>/</td><td>/=</td><td>:</td></tr>
<tr><td>;</td><td><</td><td><<</td><td><<=</td><td><=</td><td>=</td><td>==</td><td>></td><td>>=</td><td>>></td><td>>>=</td></tr>
<tr><td>?</td><td>[</td><td>]</td><td>^</td><td>^=</td><td>{</td><td>|</td><td>|=</td><td>||</td><td>}</td><td>~</td></tr>
</table>

2.3 Datentypen und Definitionen

2.3.1 Datentypen

Die **Bartels User Language** stellt die folgenden elementaren Datentypen zur Verfügung:

`char`	Zeichen aus dem Zeichensatz der Maschine
`int`	ganzzahliger numerischer Wert
`double`	doppelt genauer numerischer Gleitkommawert
`string`	Zeichenkette (**`char`**-Vektor)
`index`	Index auf definierte DDB-Struktur des Bartels AutoEngineer

Daneben ist die Verwendung folgender zusammengesetzter Datentypen möglich:

Vektor	Zusammenfassung von Elementen gleichen Datentyps
`struct`	Zusammenfassung von Elementen unterschiedlichen Datentyps

Datentyp-Konvertierung

Verschiedene Operatoren können implizite Datentypumwandlungen verursachen. So setzen eine Reihe arithmetischer Operationen definierte Datentypen bzw. Datentyppaare für ihre Operanden voraus. Ebenso wird bei der Zuweisung eines Wertes an eine Variable oder der Übergabe von Funktionsparametern eine entsprechende Datentypkompatibilität verlangt. Sofern an irgendeiner Stelle im Programm die geforderte Kompatibilität nicht vorliegt, wird versucht, den Wert des betroffenen Operanden in den gewünschten Datentyp zu überführen (man spricht von einem "type cast"). Diese Typkonvertierung läuft nach folgenden Regeln ab: Zulässige Umwandlungen ohne Informationsverlust sind **`char`** in **`int`** oder **`char`** in **`string`**, sowie **`int`** in **`double`**; ebenfalls zulässige Umwandlungen - jedoch mit möglichem Informationsverlust - sind **`int`** in **`char`**, sowie **`double`** in **`int`**. Der Compiler gibt Fehlermeldungen aus, wenn auch unter Ausnutzung der Typkonvertierungsregeln keine Typkompatibilität erreicht werden kann.

2.3.2 Variablen

Alle globalen und lokalen Variablen müssen vor ihrem Gebrauch deklariert werden, damit sowohl ihr Datentyp als auch ihr Name definiert sind. Durch derartige Vereinbarungen wird festgelegt, wie einzelne, vom Benutzer eingeführte Namen durch die **User Language** zu interpretieren sind. Jede Vereinbarung besteht aus einer Datentypspezifikation sowie einer Liste von Deklaratoren; die Deklaratoren wiederum bestehen aus dem Variablennamen sowie wahlweise einer Initialisierung der Variablen.

Elementare Datentypen

Beispiel für die Deklaration von **char**-Variablen:

```
char c;
char TAB = '\t', NEWLINE = '\n';
```

In obigem Beispiel werden die **char**-Variablen **c** (nicht initialisiert) sowie **TAB** und **NEWLINE** (initialisiert mit dem Tabulator- bzw. dem Zeilenvorschub-Zeichen) deklariert.

Beispiel für die Deklaration von **int**-Variablen:

```
int i, MAXLINELEN = 80;
int pincount = 0;
```

In obigem Beispiel werden die **int**-Variablen **i** (nicht initialisiert) und **MAXLINELEN** (initialisiert mit dem Wert 80), sowie **pincount** (initialisiert mit 0) vereinbart.

Beispiel für die Deklaration von **double**-Variablen:

```
double x_coord, y_coord;
double MMTOINCH = 1.0/25.4;
double starttime = clock();
```

In obigem Beispiel werden die **double**-Variablen **x_coord** und **y_coord** (nicht initialisiert), **MMTOINCH** (initialisiert mit einem numerischen Ausdruck), sowie **starttime** deklariert; die Variable **starttime** wird initialisiert mit dem Resultatwert der (System-)Funktion **clock**, welche die Prozessor-Zeit zurückgibt.

Beispiel für die Deklaration von **string**-Variablen:

```
string s1;
string ProgName = "TESTPROGRAM", ProgVer = "V1.0";
string ProgHeader = ProgName+"\t"+ProgVer;
```

In obigem Beispiel werden die **string**-Variablen **s1** (nicht initialisiert), **ProgName** und **ProgVer** (initialisiert mit **TESTPROGRAM** und **V1.0**), sowie **ProgHeader** deklariert; **ProgHeader** wird dabei mit einem Ausdruck initialisiert, der sich durch Aneinanderfügen der **string**-Variablen **ProgName**, des Tabulatorzeichens, sowie der **string**-Variablen

ProgVer ergibt.

Beispiel für die Deklaration von **index**-Variablen:

```
index L_MACRO macro;
index L_CNET net1, net2;
```

In obigem Beispiel werden die **index**-Variablen **macro** (vom **index**-Variablentyp **L_MACRO**) sowie **net1** und **net2** (vom **index**-Variablentyp **L_CNET**) vereinbart. Bei der Deklaration von **index**-Variablen besteht die Spezifikation des Datentyps aus dem Schlüsselwort **index** und zusätzlich dem Namen des **index**-Variablentyps (hier **L_MACRO** bzw. **L_CNET**). Die Namen für die **index**-Variablentypen sind vordefiniert (siehe Anhang B). Es muss sichergestellt sein, dass in einem Programm nur zueinander kompatible **index**-Variablentypen verwendet werden. Dies beruht auf der Tatsache, dass über **index**-Datentypen der Zugriff auf entsprechende Einträge aus der Design-Datenbank (DDB) des **Bartels AutoEngineer** definiert wird; die Verfügbarkeit dieser DDB-Einträge unterscheidet sich je nach Interpreterumgebung (im **Schaltplaneditor** sind andere Datentypen definiert als im Layout). Bei der Verwendung zueinander nicht kompatibler **index**-Variablentypen im selben Programm gibt der **User Language Compiler** eine entsprechende Fehlermeldung aus und erzeugt keinen Code. Ähnlich verhält sich der **User Language Interpreter**; beim Versuch ein **User Language**-Programm aufzurufen, das zur Interpreterumgebung inkompatible **index**-Datentyp-Referenzen enthält, gibt das System eine entsprechende Fehlermeldung aus und führt das betreffende Programm nicht aus. Die Information über die Kompatibilität der **index**-Datentypen ist dem Anhang A bzw. dem Anhang B zu entnehmen.

Vektoren

Unter einem Vektor, auch bezeichnet als Feld oder Array, versteht man die Zusammenfassung einzelner Elemente gleichen Datentyps. Bei der Deklaration von Vektorvariablen wird neben der Spezifikation des Datentyps und der Definition des Variablennamens zusätzlich die Angabe der Vektordimension benötigt. Diese Dimensions-Angabe erfolgt durch Anfügen eckiger Klammern an den Variablennamen, wobei jeweils ein Klammernpaar einer Dimension entspricht. Bei der Initialisierung von Vektor-Elementen sind die Initialisierungswerte durch Kommata zu trennen, und jede Vektordimension ist in geschweifte Klammern einzuschließen.

Beispiel für die Deklaration von Vektor-Variablen:

```
int intary[], intfield[][][];
double valtab[][] = {
      { 1.0, 2.54, 3.14 },
      { 1.0/valtab[0][1], clock() }
      };
string TECHNOLOGIES[] = {
      "TTL", "AC", "ACT", "ALS", "AS", "F",
      "H", "HC", "HCT", "HCU", "L", "LS", "S"
      };
```

Obiges Beispiel enthält die Deklarationen der **`int`**-Vektoren **`intary`** und **`intfield`** (ein- bzw. dreidimensional), des zweidimensionalen **`double`**-Vektors **`valtab`**, sowie des eindimensionalen **`string`**-Vektors **`TECHNOLOGIES`**. In den Deklarationen für **`valtab`** und **`TECHNOLOGIES`** sind Initialisierungen angegeben, die den folgenden Zuweisungen entsprechen:

```
valtab[0][0] = 1.0;
valtab[0][1] = 2.54;
valtab[0][2] = 3.14;
valtab[1][0] = 1.0/valtab[0][1];
valtab[1][1] = clock();
TECHNOLOGIES[0] = "TTL";
TECHNOLOGIES[1] = "AC";
TECHNOLOGIES[2] = "ACT";
:
TECHNOLOGIES[11] = "LS";
TECHNOLOGIES[12] = "S";
```

Es sei an dieser Stelle nochmals darauf hingewiesen, dass in der **User Language** der elementare Datentyp **`string`** einem eindimensionalen **`char`**-Vektor entspricht. Die Deklarationen

```
string s;
```

und

```
char s[];
```

sind also äquivalent.

Strukturen

Unter einer Struktur versteht man die Zusammenfassung mehrerer, jeweils durch Name und Datentyp definierter Elemente, die in einem bestimmten verarbeitungstechnischen oder auch nur logischen Zusammenhang stehen. Bei der Vereinbarung von Strukturen unterscheidet man die Strukturdefinition und die Strukturdeklaration. Die Strukturdefinition besteht aus dem Schlüsselwort **`struct`**, dem Namen der Strukturdefinition, und - in geschweiften Klammern - den Definitionen der Strukturelemente. Die Strukturdeklaration besteht aus dem Schlüsselwort **`struct`**, dem Namen einer gültigen Strukturdefinition, sowie dem Namen der Variablen, die der Strukturdefinition zugeordnet wird. Strukturdefinition und Strukturdeklaration können zusammengefasst werden, wobei dann auch der Name für die Strukturdefinition entfallen kann. Initialisierungen innerhalb von Strukturdeklarationen sind in der aus den Vektordeklarationen bekannten Nomenklatur zulässig.

Beispiel für die Vereinbarung von Strukturen:

```
// Structure declarations

struct coordpair {
      double x
      double y;
      };

struct coordpair elementsize = {
      bae_planwsux()-bae_planwslx(),
      bae_planwsuy()-bae_planwsly()
      };

struct elementdes {
      string fname, ename;
      int class;
      struct coordpair origin, size;
      } element = {
              bae_planfname(),
              bae_planename(),
              bae_planddbclass(),
              {
                      bae_planwsnx(),
                      bae_planwsny()
                      },
              elementsize
              };

struct {
      string id, version;
      struct {
              int day;
              string month;
              int year;
              } reldate;
      } program = {
              "UL PROGRAM",
              "Version 1.1",
              { 4, "July", 1992 }
              };
```

Obiges Beispiel enthält die Definition der Struktur **`coordpair`**, die Deklaration der Variablen **`elementsize`** (Struktur vom Typ **`coordpair`**), die Definition der Struktur **`elementdes`**, die Deklaration der Variablen **`element`** (Struktur vom Typ **`elementdes`**), sowie die Deklaration der Strukturvariablen **`program`**. Die in den Deklarationen für **`elementsize`**, **`element`** und **`program`** enthaltenen Initialisierungen entsprechen den folgenden Zuweisungen:

```
elementsize.x=bae_planwsux()-bae_planwslx();
elementsize.y=bae_planwsuy()-bae_planwsly();
element.fname=bae_planfname();
element.ename=bae_planename();
element.class=bae_planddbclass();
element.origin.x=bae_planwsnx();
element.origin.y=bae_planwsny();
element.size=plansize;
program.id="UL PROG";
program.version="Version 1.1";
program.reldate.day=4;
program.reldate.month="July";
program.reldate.year=1992;
```

Auch die Definition von Vektoren aus Strukturen bzw. die Verwendung von Vektordatentypen innerhalb von Strukturen ist möglich, wie das folgende Beispiel demonstriert:

```
struct drilldef {
      index L_DRILL drilltool;
      struct { double x, y; } drillcoords[];
      } drilltable[];
```

Datentypumbenennung

Bartels User Language verfügt über einen Mechanismus zur Umbenennung von Datentypen. Dabei handelt es sich nicht um die Schaffung eines neuen Datentyps, sondern lediglich um die Vergabe eines zusätzlichen Namens für einen bereits bekannten Typ. Eine derartige Typumbennung erfolgt durch die Angabe des Schlüsselwortes **`typedef`** gefolgt von der Spezifikation des Datentyps sowie dem zur Bezeichnung dieses Datentyps zusätzlich einzuführenden Namen. Ein mit **`typedef`** eingeführter Name kann anschließend zur Typspezifikation bei der Deklaration von Variablen, Funktionen und Funktionsparametern verwendet werden.

Beispiel für Typ-Umbennungen:

```
typedef index L_CNET NETLIST[];
typedef int IARY[];
typedef IARY MAT_2[];
typedef struct {
      int pointcount;
      struct {
              int t;
              double x,y;
              } pointlist[];
      } POLYLIST[];
MAT_2 routmatrix;
NETLIST netlist;
POLYLIST polygonlist;
```

In obigem Beispiel werden die drei Variablen **`routmatrix`** (zweidimensionaler **`int`**-Vektor), **`netlist`** (eindimensionaler **`index`**-Vektor vom Typ **L_CNET**), sowie **`polygonlist`** (eindimensionaler Vektor aus Strukturen, die ihrerseits ein **`int`**-Element und einen **`struct`**-Vektor enthalten) deklariert.

2.3.3 Funktionen

Bei komplexen Operationen und Berechnungen ist es meist nicht notwendig, zu wissen, wie das Ergebnis zustande kommt; interessant ist lediglich das Ergebnis selbst. Auch ist es wünschenswert, bestimmte Bearbeitungsfolgen immer wieder verwenden zu können. Mit Hilfe von Funktionen ist die Zerlegung großer Problemstellungen in kleinere, d.h. die Modularisierung und Vereinfachung von Programmen möglich. In der **Bartels User Language** wird unterschieden zwischen den Systemfunktionen und den durch den Anwender definierten Funktionen.

Funktionsdefinition

Die Definitionen der Systemfunktionen sind dem Compiler bekannt, die Funktionen selbst sind in den Interpreter eingebunden. Anhang C enthält die Beschreibung dieser Systemfunktionen. Der Anwender kann also bei der Implementierung seiner Programme die Systemfunktionen verwenden und hat darüber hinaus die Möglichkeit, eigene Funktionen zu definieren.

Eine Funktionsdefinition besteht aus dem Funktionskopf (Header) und dem Funktionsrumpf (Block). Der Header enthält eine Typspezifikation und den Namen der Funktion, sowie die Definition und Deklaration der Funktionsparameter. Die Typspezifikation definiert den Datentyp des Wertes, den die Funktion an den Aufrufer zurückliefert (Rückgabewert). Hierbei steht zusätzlich der Datentyp **`void`** zur Verfügung, der dem Compiler angibt, dass die Funktion keinen Rückgabewert liefert. Wird die Datentypspezifikation weggelassen, dann wird der Funktion automatisch der Datentyp **`int`** zugeordnet. Nach dem Funktionsnamen folgt die Parameterdefinition. Diese besteht aus einer in runde Klammern eingeschlossenen Liste von durch Kommata getrennten Parameternamen bzw. Parameterdeklarationen. Für den Fall, dass die Funktion gar keine Parameter enthält, ist lediglich das Klammernpaar anzugeben. Alle definierten Parameter (außer den bereits in der Liste der Parameterdefinitionen deklarierten sowie den **`int`**-Typen) müssen nach der Liste der Parameterdefinitionen explizit vereinbart werden. Diese Parameterdeklarationen sind wie Variablendeklarationen vorzunehmen. Nach dem Funktionskopf muss der Funktionsrumpf definiert werden. Dieser ist ein durch geschweifte Klammern umschlossener Block von Anweisungen.

Beispiele für Funktionsdefinitionen:

```
double netroutwidth(index L_CNET net)
// Get the routing width of a given net
// Returns : width or 0.0 if two pins with different width
{
      index L_CPIN pin;       // Pin index
      int pincnt=0;           // Pin count
      double rw=0.0;          // Rout width
      // Loop thru all pins
      forall (pin of net) {
              // Test if the pin introduces a new rout width
              if (pin.RWIDTH!=rw && pincnt++>0)
                      return(0.0);
              // Set the rout width
              rw=pin.RWIDTH;
              }
      // Return the rout width
      return(rw);
}

int allpartsplaced()
// Test if all netlist-defined parts are placed
// Returns : 1 if all parts are placed or zero otherwise
{
      index L_CPART cpart;    // Connection part index
      // Loop thru the connection part list
      forall (cpart where !cpart.USED)
              // Unplaced part matched
              return(0);
      // All parts are placed
      return(1);
}
```

```
double getdistance(xs,ys,xe,ye)
// Get the distance between two points
// Returns : the distance length value
double xs, ys;                // Start point coordinate
double xe, ye;                // End point coordinate
{
      double xd=xe-xs;        // X distance
      double yd=ye-ys;        // Y distance
      // Calculate and return the distance (Pythagoras)
      return(sqrt(xd*xd+yd*yd));
}

double arclength(r,a1,a2)
// Get arc segment length by radius and start-/end-point angle
// Returns : the arc segment length value
double r;                     // Radius
double a1;                    // Start point angle (in radians)
double a2;                    // End point angle (in radians)
{
      // Arc; "absolute" angle between start and end point
      double arc = a1<a2 ? a2-a1 : 2*PI()+a2-a1;
      // Get and return the arc segment length
      return(arc*r);
}

double getangle(cx,cy,x,y)
// Get the angle of a circle arc point
// Returns : the angle (in radians; range [0,2*PI])
double cx, cy;                // Circle center coordinate
double x, y;                  // Circle arc point coordinate
{
      double res;             // Result value
      // Get arc tangent of angle defined by circle point
      res=atan2(y-cy,x-cx);
      // Test the result
      if (res<0.0)
              // Get the "absolute" angle value
              res=PI()-res;
      // Return the result value
      return(res);
}

double PI()
// Returns the value of PI in radians
{
      // Convert 180 degree and return the result
      return(cvtangle(180.0,1,2));
}

void cputimeuse(rn,st)
// Report CPU time usage (in seconds)
string rn;                    // Routine name
double st;                    // Start time
{
      // Print CPU time elapsed since start time
      printf("(%s) Elapsed CPU Time = %6.1f [Sec]\n",rn,clock()-st);
}
```

Funktionsaufruf und Wertübergabe

Jede in einem **User Language**-Programm (bzw. Programmtext) per Definition bekannte Funktion kann innerhalb dieses Programmes (bzw. in dem entsprechenden Programm-Modul) auch aufgerufen werden. Bei der Verwendung von **User Language** Systemfunktionen besteht jedoch die Einschränkung, dass in einem Programm nur zueinander kompatible Systemfunktionen verwendet werden können. Dies beruht auf der Tatsache, dass über derartige Funktionsaufrufe ggf. Aktionen ausgelöst werden, die nur in einer bestimmten Interpreterumgebung ausgeführt werden können (die Funktion zur Festlegung von Plotparametern im **CAM-Prozessor** des **AutoEngineers** kann selbstverständlich nicht im **Schaltplaneditor** aufgerufen werden). Bei der Verwendung zueinander nicht kompatibler Systemfunktionen gibt der **User Language Compiler** eine Fehlermeldung aus und erzeugt keinen Programm-Code. Ähnlich verhält sich der **User Language Interpreter**; beim Versuch ein **User Language**-Programm aufzurufen, das zur Interpreterumgebung inkompatible Systemfunktions-Referenzen enthält, erzeugt das System eine entsprechende Fehlermeldung und führt das betreffende Programm nicht aus. Die Information über die Kompatibilität der **User Language**-Systemfunktionen ist dem Anhang A bzw. dem Anhang C zu entnehmen.

Der Funktionsaufruf setzt sich zusammen aus dem Funktionsnamen und der in runden Klammern eingeschlossenen Liste der aktuellen Parameter, die der Funktion übergeben werden soll.

Die Inhalte der global definierten Variablen eines Programms stehen grundsätzlich in jeder Funktion desselben Geltungsbereichs zur Verfügung, d.h. globale Variablen können zur Übergabe von Werten an Funktionen benutzt werden. Darüber hinaus besteht die Möglichkeit der Wertübergabe über Funktionsparameter. Die Übergabe per Parameter kann einfach kontrolliert werden und ist daher i.d.R. der Methode der Übergabe über globale Variablen vorzuziehen. Die Liste der aktuellen Parameter, also der Ausdrücke, die man bei einem Funktionsaufruf übergibt, muss mit der formalen Parameterdefinition (also Anzahl und Datentypen) der aufzurufenden Funktion übereinstimmen. Beim Funktionsaufruf werden die Werte der aktuellen Parameter in die entsprechenden formalen Parameter kopiert. Nach erfolgreicher Beendigung der Funktion wird jeder durch die Funktion geänderte Parameterwert wieder auf den aktuellen Parameter zurückgespeichert, sofern dieser eine Variablenreferenz darstellt. Schließlich besteht noch die Möglichkeit der Wertübergabe über den Rückgabewert der Funktion. Dabei wird innerhalb der Funktion mit Hilfe der **`return`**-Anweisung ein Funktionsergebnis gesetzt, das anschließend vom Aufrufer innerhalb des Ausdrucks, der den Funktionsaufruf enthält, ausgewertet werden kann.

Beispiel für Funktionsaufruf und Wertübergabe:

```
// Date structure
struct date { int day, month, year; };
// Global program variables
string globalstr="Global string";
int fctcallcount=0;

// Main program
main()
{
      // Local variables of main
      string resultstr="function not yet called";
      struct date today = { 0, 0, 0 };
      double p=0.0, b=2.0, e=10.0;
      // Print the global variables
      printf("fctcallcount=%d, %s\n",fctcallcount,globalstr);
      // Print the local variables
      printf("resultstr=\"%s\"\n",resultstr);
      printf("today : %d,%d,%d",today.day,today.month,today.year);
      printf("\t\tb=%.1f, e=%.1f, p=%.1f\n",b,e,p);
      // Call function
      resultstr=function(today,b,e,p);
      // Print the global variables
      printf("fctcallcount=%d, %s\n",fctcallcount,globalstr);
      // Print the local variables
      printf("resultstr=\"%s\"\n",resultstr);
      printf("today : %d,%d,%d",today.day,today.month,today.year);
      printf("\t\tb=%.1f, e=%.1f, p=%.1f\n",b,e,p);
}
```

```
string function(curdate,base,exponent,power)
struct date curdate;          // Current date parameter
double base;                  // Base parameter
double exponent;              // Exponent parameter
double power;                 // Power parameter
{
      // Increment the function call count
      fctcallcount++;
      // Set the global string
      globalstr="Global string changed by function";
      // Get the current date
      get_date(curdate.day,curdate.month,curdate.year);
      // Calculate the power
      power=pow(base,exponent);
      // Return with a result string
      return("function result string");
}
```

Obiges Beispiel-Programm erzeugt folgende Ausgabe:

```
fctcallcount=0, Global string
resultstr="function not yet called"
today : 0,0,0         b=2.0, e=10.0, p=0.0
fctcallcount=1, Global string changed by function
resultstr="function result string"
today : 4,6,92                b=2.0, e=10.0, p=1024.0
```

Kontrollfluss und Programmstruktur

Jede Funktion der **User Language** behält nach ihrem Aufruf solange die Kontrolle, bis sie auf einen weiteren Funktionsaufruf, auf eine **`return`**-Anweisung, oder nach der Abarbeitung der letzten Anweisung der Funktion auf das Funktionsende trifft. Bei einem Funktionsaufruf wird die Kontrolle an die aufgerufene Funktion weitergegeben. Beim Erreichen einer **`return`**-Anweisung oder am Funktionsende wird die Kontrolle an den Aufrufer der Funktion zurückgegeben. Ist in einem Programm eine Anwenderfunktion mit dem Namen **`main`** definiert, dann erzeugt der **User Language Compiler** Programm-Code, der dafür sorgt, dass - unmittelbar nach der Initialisierung der globalen Variablen - diese Funktion aufgerufen wird. Der Kontrollfluss eines **User Language**-Programms beginnt also üblicherweise bei der Funktion **`main`**. Da bei rekursionsfreier Programmierung jede Funktion irgendwann ihre Kontrolle wieder an den Aufrufer zurückgibt, fällt die Kontrolle schließlich wieder an die **`main`**-Funktion. Wird darin dann das Funktionsende oder eine **`return`**-Anweisung erreicht, dann ist auch das Programmende erreicht, und der **User Language Interpreter** kann den Kontrollfluss beenden.

Rekursive Funktionen

Funktionen dürfen rekursiv benutzt werden, d.h. eine Funktion darf sich direkt oder indirekt selbst aufrufen. Dies ist jedoch nur dann sinnvoll, wenn sich bei jedem Aufruf ein Zustand derart verändert, dass sich irgendwann ein eindeutig definierter Endzustand einstellt, mit dessen Hilfe sich die Rekursion abbrechen lässt.

Durch die rekursive Programmierung von Funktionen kann man im Allgemeinen Programm-Code einsparen und eventuell die Lesbarkeit erhöhen. Die Programmlaufzeit und der Speicherplatzbedarf hingegen werden sich durch Rekursionen erhöhen. Daher sollte man jeweils prüfen, ob die Verwendung einer Rekursion tatsächlich nützlich ist. Auch besteht die Gefahr, dass man "versehentlich" eine Endlosrekursion implementiert, d.h. eine Rekursion, die nie den Endzustand erreicht. Der **User Language Interpreter** wird bei der Abarbeitung einer solchen Endlosrekursion irgendwann einen Speicher- bzw. Stapel-Überlauf feststellen, da für jeden Funktionsaufruf zumindest eine Rücksprung-Adresse gespeichert werden muss.

2.3.4 Regeln zum Geltungsbereich

Bei der Referenzierung der Objekte eines **User Language**-Programms muss der Compiler jeweils die Gültigkeit der Referenz überprüfen. Hierzu ist jedem Objekt ein bestimmter Geltungsbereich innerhalb des Programms zugeordnet. Innerhalb dieses Geltungsbereiches ist das entsprechende Objekt bekannt und kann referenziert werden. Es wird unterschieden zwischen globalem und lokalem Geltungsbereich. Der globale Geltungsbereich erstreckt sich über das gesamte Programm (also auch auf noch einzubindende, getrennt kompilierte Programmteile bzw. Libraries), während die lokalen Geltungsbereiche des Programms Bezug nehmen auf die Funktionsdefinitionen.

Die Funktionen eines Programms sind global, d.h. im gesamten Programm gültig. Variablen- und Typ-Definitionen, die in einer Funktion vorgenommen werden, gelten nur lokal innerhalb dieser Funktion; außerhalb von Funktionen gelten derartige Definitionen als global. Die Parameterdefinitionen einer Funktione werden wie lokale Variablen behandelt und gelten daher immer nur lokal innerhalb der betreffenden Funktion. Strukturdefinitionen gelten allgemein als global im aktuell zu übersetzenden Programmtext. Durch die Zuweisung der Speicherklasse **`static`** kann der Geltungsbereich von Funktionen und globalen Variablen eingeschränkt werden auf den aktuell zu übersetzenden Programmtext. Die Speicherklasse **`static`** dient insbesondere der Vermeidung von Namenskonflikten beim Binden bzw. Linken unterschiedlicher Programm-Module bzw. Libraries.

Um Namenskonflikte zu vermeiden, müssen die Elemente jeder Objektklasse innerhalb ihres Geltungsbereichs jeweils unterschiedliche Namen besitzen. Bei der Referenzierung haben die lokalen Objekte Vorrang vor den globalen.

2.4 Ausdrücke

In diesem Abschnitt werden die in der **User Language** verfügbaren Operatoren für Ausdrücke vorgestellt. Die Reihenfolge, in der dabei vorgegangen wird, entspricht dem abnehmenden Vorrang der Operatoren, wobei jeweils auch auf die Assoziativität d.h. die Verarbeitungsfolge der jeweiligen Operation eingegangen wird. Jeder Operator ist entweder links- oder rechts-assoziativ, je nachdem ob er von links oder rechts implizit geklammert wird. Der Vorrang gibt an, mit welcher Priorität ein Ausdruck bewertet wird. Abgesehen vom Vorrang ist die Reihenfolge der Bewertung undefiniert. Es ist der Implementierung freigestellt, Teilausdrücke auch dann, wenn diese Nebeneffekte verursachen, in möglichst effizienter Reihenfolge zu bewerten. Die Reihenfolge für die Evaluierung von Nebeneffekten ist dabei nicht definiert. Ausdrücke mit assoziativen und kommutativen Operatoren lassen sich beliebig umordnen. Eine bestimmte Reihenfolge in der Bewertung von Operanden kann nur durch Zuweisung an (temporäre) Variablen erzwungen werden.

2.4.1 Primäre Ausdrücke

Primäre Ausdrücke, auch bezeichnet als einfache Ausdrücke oder Primary, sind Konstanten, Verweise auf Objekte, Ausdrucks-Klammerungen, Funktionsaufrufe, sowie die Auswahl von Elementen aus einem Vektor, einer Struktur oder einem Index. Diese Operationen sind links-assoziativ.

Konstanten und Verweise auf Objekte

Jeder geeignet vereinbarte Identifier gilt als primärer Ausdruck, der auf ein Objekt verweist. Auch ganzzahlige, Gleitkomma-, Zeichen- und String-Konstanten sind primäre Ausdrücke; die Darstellungsformen für Konstanten wurden in Kapitel 2.2.3 vorgestellt.

Ausdrucks-Klammerung

Ein geklammerter Ausdruck besteht aus einem in runde Klammern eingeschlossenen Ausdruck und stellt einen primären Ausdruck dar. Mit Hilfe dieses Operators können Ausdrücke explizit geklammert werden, wodurch eine definierte Änderung der Reihenfolge der Bewertung herbeigeführt werden kann. Da der Multiplikations-Operator z.B. höheren Vorrang hat als der Additions-Operator, ergibt sich der Wert des Ausdrucks

```
a + b * c
```

aus der Summe aus a und dem Produkt b*c, während er sich in

```
(a + b) * c
```

aus dem Produkt aus der Summe a+b und c ergibt.

Funktionsaufruf

Ein Funktionsaufruf besteht aus einem Identifier zur Bezeichnung der aufzurufenden Funktion gefolgt von einer in runde Klammern eingeschlossenen Liste von Ausdrücken, die durch Kommata getrennt sind. Die Werte der Ausdrücke innerhalb der runden Klammern repräsentieren die aktuellen Parameter für den Funktionsaufruf. Sofern eine Funktion per Definition einen Resultatwert liefert, kann der entsprechende Funktionsaufruf in einem beliebigen anderen Ausdruck verwendet werden. Die Liste der Ausdrücke innerhalb der runden Klammern kann leer sein. Typische Funktionsaufrufe sind z.B.

```
init_states(pass=1);
printf("This is a message!\n");
printf("Element %s\n",bae_planename());
xpos=nref_xcoord(ask_partname())-bae_planwsnx();
```

Zugriff auf Vektor-Element

Ein einfacher Ausdruck gefolgt von einem Ausdruck in eckigen Klammern ist wiederum ein einfacher Ausdruck. Mit dieser Operation erfolgt die Auswahl eines Elements aus einem Vektor oder **`string`**-Wert. Der Ausdruck vor der eckigen Klammer referenziert dabei den Vektor, der Ausdruck innerhalb der eckigen Klammern wird als **`int`**-Wert interpretiert und dient der Indizierung des Vektor-Elements. Der Indizierungs-Wert darf dabei nicht negativ sein; der Wert 0 (Null) verweist auf das erste Element des Vektors. Beim Speichern auf einen Vektor erfolgt ggf. eine dynamische Anpassung des Vektor-Feldbereichs durch den **User Language Interpreter**. Der lesende Zugriff auf einen Vektor ist nur innerhalb des aktuell definierten Feldbereiches zulässig. Die folgende Funktion **`strisoctal`** überprüft, ob der als Parameter übergebene **`string`**-Wert nur oktale Ziffern (**`0`** bis **`7`**) enthält und gibt für diesen Fall den Wert 1, in allen anderen Fällen den Wert 0 zurück:

```
int strisoctal(string str)
{
      for (i=0;i<strlen(str);i++)
              if (!isdigit(str[i]) || str[i]=='8' || str[i]=='9')
                      return(0);
      return(1);
}
```

In obigem Beispiel wird die Feldlängen-Kontrolle für den Vektor-Lese-Zugriff mit Hilfe der auf **`string`**-Werte anwendbaren Systemfunktion **strlen** durchgeführt. Im folgenden Programm-Konstrukt hingegen geschieht dies mit Hilfe einer speziellen **`int`**-Variablen (**`filecount`**):

```
string curfilename="", filelist[];
int i, filecount=0;
while (scandirfnames(".",".ddb",curfilename)==1)
      filelist[filecount++]=curfilename;
for (i=0;i<filecount;i++)
      printf("File %s \n",filelist[i]);
```

In obigem Beispiel wird zunächst eine Liste aller im aktuellen Verzeichnis auffindbaren Dateinamen mit der Endung **`.ddb`** erzeugt, und anschließend wird diese Liste ausgegeben.

Zugriff auf Struktur- oder Index-Element

Ein einfacher Ausdruck gefolgt von einem Punkt und einem Identifier ist wieder ein einfacher Ausdruck. Der erste Ausdruck referenziert dabei eine Struktur bzw. einen Index-Typ, und der Identifier nach dem Punkt-Operator muss ein definiertes Element aus dieser Struktur bzw. diesem Index-Typ bezeichnen. Der schreibende Zugriff auf Struktur-Elemente ist grundsätzlich immer möglich; auf die Elemente von Index-Typen hingegen darf nicht gespeichert werden (Fehlermeldung durch den Compiler). Umgekehrt ist der lesende Zugriff auf die Elemente einer aktuell gültigen **`index`**-Variablen immer zulässig, während aus **`struct`**-Variablen nur die zuvor initialisierten gelesen werden können (ansonsten Speicherzugriffs-Verletzung und entsprechende Fehlermeldung durch den Interpreter). Der folgende Programmteil definiert eine Liste bestehend aus Strukturen und erzeugt für jedes aktuell verfügbare Layout-Macro einen Listen-Eintrag mit dem Macro-Namen und der Macro-Klasse:

```
int macrocnt=0;
struct { string name; int class; } macrolist[];
index L_MACRO macro;
forall (macro) {
      macrolist[macrocnt].name=macro.NAME;
      macrolist[macrocnt++].class=macro.CLASS;
      }
```

2.4.2 Unitäre Ausdrücke

Unitäre Ausdrücke, auch bezeichnet als Unary, sind rechts-assoziativ und umfassen alle Operatoren, die einen einzelnen Operanden bewerten.

Inkrement und Dekrement

Die Operatoren **++** bzw. **--** verändern ihre Operanden, indem sie den Wert 1 zum Wert ihres Operanden addieren bzw. subtrahieren. Sind diese Operatoren einem unitären Ausdruck vorangestellt, dann entspricht das Resultat dem Wert des Operanden nach dessen Inkrementierung bzw. Dekrementierung. Im anderen Fall, also wenn **++** bzw. **--** auf den Operanden folgt, dann wird zwar der Operand inkrementiert bzw. dekrementiert, das Resultat entspricht jedoch dem Wert des Operanden vor der Durchführung der Operation. Dies bedeutet, dass in einem Kontext, in dem der Resultatwert einer Inkrement- bzw. Dekrement-Operation weiter verwendet wird, die entsprechenden Ausdrücke unterschiedliche Bedeutung haben. Besitzt zum Beispiel die Variable **count** den Wert 12 dann erhält die Variable **n** durch die Zuweisung

```
n = count-- ;
```

den Wert 12, durch die Zuweisung

```
n = --count ;
```

jedoch den Wert 11 (der Wert von **count** ergibt sich in beiden Fällen zu 11).

Arithmetische Negation

Der unitäre Operator **-** liefert den Wert der arithmetischen Negation seines Operanden, das Resultat ist also der mit (-1) multiplizierte Wert des Operanden.

Logische Negation

Der unitäre Operator **!** liefert den Wert der logischen Negation seines Operanden. Das Resultat ist dabei 1 für einen Operanden mit dem Null-Wert (0 oder Leerstring für **string**-Operand), und 0 für alle anderen Operanden-Werte.

Bit-Komplement

Der Operator **~** komplementiert die einzelnen Bits in seinem Operanden, liefert also als Ergebnis das 1-Komplement seines Operanden.

2.4.3 Binäre Ausdrücke

Binäre Ausdrücke, auch bezeichnet als Binary, umfassen alle Operatoren, die zwei Operanden bewerten.

Produkt

Die Operatoren für Multiplikation und Division sind links-assoziativ und erzeugen jeweils einen Produkt-Ausdruck. Der binäre Operator **`*`** ist kommutativ und assoziativ und bezeichnet die Multiplikation. Das Ergebnis dieser Operation ist der Wert des Produktes seiner beiden Operanden. Der binäre Operator **`/`** bezeichnet die Division. Das Ergebnis dieser Operation ist der Wert, der durch die Division des Wertes des ersten Operanden (Divident) geteilt durch den Wert des zweiten Operanden (Divisor) der Operation zustande kommt. Da eine Division durch Null nicht zulässig ist, ist der Wert 0 für den Divisor auch nicht erlaubt. Der binäre Operator **`%`** liefert den Rest nach Division seiner beiden Operanden. Gleitkomma-Operanden sind hierbei nicht zulässig; ebenso ist der Wert 0 für den Divisor nicht erlaubt; ein Beispiel für die Verwendung des **`%`**-Operators ist der Ausdruck

```
febdays = (year%4==0 && year%100!=0 || year%400==0) ? 29 : 28 ;
```

in dem zunächst überprüft wird, ob der Wert von **`year`** der Jahreszahl eines Schaltjahres entspricht, und anschließend in Abhängigkeit vom Ergebnis dieser Überprüfung der Wert der Variablen **`febdays`** gesetzt wird.

Summe

Die Operatoren für Addition und Subtraktion sind links-assoziativ und erzeugen jeweils einen Summen-Ausdruck. Der binäre Operator **`+`** bezeichnet die Addition. Für den Fall, dass dieser Operator auf numerische Werte angewendet wird, ist das Ergebnis dieser Operation die Summe der Operanden, und die Operation ist in diesem Fall assoziativ und kommutativ; wird diese Operation auf Zeichenketten angewendet, dann ergibt sich das Resultat zu dem **`string`**-Wert, der sich durch Aneinanderfügen der **`string`**-Werte der Operanden ergibt. Der binäre Operator **`-`** bezeichnet die Subtraktion und liefert als Ergebnis den Wert der Differenz seiner Operanden.

Schiebe-Operation

Die Schiebe-Operatoren **`<<`** und **`>>`** sind links-assoziativ und können auf ganzzahlige Operanden angewendet werden. Der binäre Operator **`<<`** bezeichnet die bitweise Links-Verschiebung. Das Ergebnis dieser Operation ist der Wert, der sich ergibt, wenn das Bitmuster des ersten Operanden um die Anzahl der durch den zweiten Operanden gegebenen Stellen nach links verschoben wird (von rechts her werden dabei Bits mit dem Wert 0 nachgeschoben; das Ergebnis ist undefiniert, wenn der zweite Operand negativ ist). Der binäre Operator **`>>`** bezeichnet die bitweise Rechts-Verschiebung. Das Ergebnis dieser Operation ist der Wert, der sich ergibt, wenn das Bitmuster des ersten Operanden um die Anzahl der durch den zweiten Operanden gegebenen Stellen nach rechts verschoben wird (das Ergebnis ist undefiniert, wenn der zweite Operand negativ ist).

Vergleich

Die binären Operatoren **`<`** (kleiner), **`<=`** (kleiner oder gleich), **`>`** (größer) und **`>=`** (größer oder gleich) liefern den **`int`**-Wert 0, wenn die angegebene Relation in Bezug auf die beiden Operanden falsch ist, und den Wert 1, wenn die Relation vorliegt. Diese Operatoren sind auch direkt auf **`string`**-Typen anwendbar.

Äquivalenz-Vergleich

Die binären Operatoren **`==`** (gleich) und **`!=`** (ungleich) liefern den **`int`**-Wert 0, wenn die angegebene Äquivalenz-Relation in Bezug auf die beiden Operanden falsch ist, und den Wert 1, wenn die Relation vorliegt. Die Äquivalenz-Vergleichs-Operatoren sind auch direkt auf **`string`**-Typen anwendbar; sie verhalten sich analog zu den anderen Vergleichs-Operatoren, haben jedoch geringeren Vorrang.

Bitweise Und-Verknüpfung

Der binäre Operator **`&`** bezeichnet die bitweise Und-Verknüpfung (AND) und ist assoziativ und kommutativ. Das Resultat dieser Operation ist der Wert, der sich ergibt, wenn die Bits der beiden Operanden in jeder Position der Und-Verknüpfung unterworfen werden.

Bitweise Exklusiv-Oder-Verknüpfung

Der binäre Operator **^** bezeichnet die bitweise Exklusiv-Oder-Verknüpfung (XOR) und ist assoziativ und kommutativ. Das Resultat dieser Operation ist der Wert, der sich ergibt, wenn die Bits der beiden Operanden in jeder Position der Exklusiv-Oder-Verknüpfung unterworfen werden.

Bitweise Oder-Verknüpfung

Der binäre Operator **|** bezeichnet die bitweise Oder-Verknüpfung (OR) und ist assoziativ und kommutativ. Das Resultat dieser Operation ist der Wert, der sich ergibt, wenn die Bits der beiden Operanden in jeder Position der Oder-Verknüpfung unterworfen werden.

Logische Und-Verknüpfung

Der binäre Operator **&&** bezeichnet die logische Und-Verknüpfung (AND) und ist links-assoziativ. Das Resultat dieser Operation ist der **int**-Wert 1, wenn die Werte beider Operanden vom Null-Wert (0 oder Leerstring für **string**-Operanden) verschieden sind; im anderen Fall ergibt sich das Resultat zum Wert 0. Die logische Und-Verknüpfung wird strikt von links nach rechts durchgeführt; der rechte Operand wird dabei nur bewertet, wenn der linke Operand kein Null-Wert ist. Im Ausdruck

```
x<100 && fct(x)
```

wird die Funktion **fct** nur dann aufgerufen, wenn der Wert von **x** kleiner als 100 ist.

Logische Oder-Verknüpfung

Der binäre Operator **||** bezeichnet die logische Oder-Verknüpfung (OR) und ist links-assoziativ. Das Resultat dieser Operation ist der **int**-Wert 1, wenn einer der beiden Operanden-Wert vom Null-Wert (0 oder Leerstring für **string**-Operanden) verschieden ist; im anderen Fall ergibt sich das Resultat zum Wert 0. Die logische Oder-Verknüpfung wird strikt von links nach rechts durchgeführt; der rechte Operand wird dabei nur bewertet, wenn der Wert des linken Operanden dem Null-Wert entspricht. Im Ausdruck

```
test1() || test2()
```

wird die Funktion **test2** also nur dann aufgerufen, wenn zuvor der Aufruf der Funktion **test1** das Resultat 0 geliefert hat.

Bedingte Bewertung

Der Operator **?:** bezeichnet die bedingte Bewertung (conditional) und ist rechts-assoziativ. Vor dem **?:**-Operator hat ein binärer Ausdruck zu stehen, der in jedem Fall bewertet wird. Ist der Wert dieses Ausdrucks vom Null-Wert (0 oder Leerstring für **string**) verschieden, dann ergibt sich das Resultat dieser Operation zum Wert des zweiten Ausdrucks (dieser befindet sich zwischen dem Fragezeichen und dem Doppelpunkt des Operators und kann wiederum eine bedingte Bewertung sein); im anderen Fall, also wenn der erste Operand dem Null-Wert entspricht, dann ergibt sich das Resultat zum Wert des dritten Ausdrucks (dieser folgt auf den Doppelpunkt Fragezeichen des Operators und kann ebenfalls eine bedingte Bewertung sein). Der allgemeine Ausdruck

```
result = logexpr ? trueexpr : falsexpr ;
```

für die Zuweisung einer bedingte Bewertung an ein Resultat entspricht also der folgenden Kontrollstruktur:

```
if (logexpr)
      result = trueexpr ;
else
      result = falseexpr ;
```

Beispiel für die Verwendung des **?:**-Operators zur Berechnung des Maximums zweier Werte:

```
maxval = (val1>=val2) ? val1 : val2 ;
```

Zuweisungen

Der binäre Operator **`=`** bezeichnet die einfache Zuweisung, die binären Operatoren **`*=`**, **`/=`**, **`%=`**, **`+=`**, **`-=`**, **`>>=`**, **`<<=`**, **`&=`**, **`^=`** und **`|=`** bezeichnen die zusammengesetzten Zuweisungen. Alle diese Operatoren sind rechts-assoziativ. Der linke Operand muss immer ein unitärer Ausdruck sein, der rechte Operand darf seinerseits ein Zuweisungs-Ausdruck sein. Bei der einfachen Zuweisung ersetzt der Wert des rechten Operanden den Wert des durch den linken Operanden referenzierten Objekts. Eine zusammengesetzte Zuweisung der allgemeinen Form

```
expr1 <operator>= expr2
```

entspricht dem Ausdruck

```
expr1 = expr1 <operator> (expr2)
```

wobei jedoch expr1 nur einmal bewertet wird (man beachte die Klammerung des Ausdrucks expr2). In der Zuweisungsfolge

```
a = 5 ;
b = 3-a ;
c = b+a ;
c -= a *= b += 4+2*a ;
```

ergeben sich die Werte der Variablen **a**, **b** und **c** zu 60, 12 und -57.

2.4.4 Liste von Ausdrücken

Jeder Ausdruck kann aus einer Liste von durch Kommata getrennten binären Ausdrücken bestehen. Zwei Ausdrücke, die durch Komma getrennt sind, werden dabei von links nach rechts bewertet; der Wert des linken Ausdrucks wird berechnet, aber nicht weiter verwendet. Typ und Wert des Resultats einer Liste von Ausdrücken entsprechen dem Typ und Wert des rechten Ausdrucks innerhalb der Ausdruck-Liste. Diese Operation ist links-assoziativ. In einem Kontext, in dem das Komma eine spezielle Bedeutung hat, wie etwa in der Liste von Argumenten für einen Funktionsaufruf, oder in einer Liste von Initialisierungen, kann der hier beschriebene Komma-Operator nur innerhalb von Klammern verwendet werden. Durch die Zuweisung

```
c -= (a=5, b=3-a, c=b+a, a*=b+=4+2*a) ;
```

ergeben sich die Werte der Variablen **a**, **b** und **c** zu 60, 12 und -57.

2.4.5 Vorrang und Reihenfolge der Bewertung

In Tabelle 2-4 sind nochmals der Vorrang und die Assoziativität der Operatoren der **Bartels User Language** zusammengefasst dargestellt.

Tabelle 2-4: Operator Vorrang und Assoziativität

<table>
<tr><th>Operation</th><th>Operator(en)</th><th>Verarbeitungsfolge</th></tr>
<tr><td>Primary</td><td>() [] .</td><td>links nach rechts</td></tr>
<tr><td>Unary</td><td>! ~ ++ -- -</td><td>rechts nach links</td></tr>
<tr><td>Product</td><td>* / %</td><td>links nach rechts</td></tr>
<tr><td>Sum</td><td>+ -</td><td>links nach rechts</td></tr>
<tr><td>Shift</td><td><< >></td><td>links nach rechts</td></tr>
<tr><td>Comparison</td><td>< <= > >=</td><td>links nach rechts</td></tr>
<tr><td>Equality</td><td>== !=</td><td>links nach rechts</td></tr>
<tr><td>Bit And</td><td>&</td><td>links nach rechts</td></tr>
<tr><td>Bit Xor</td><td>^</td><td>links nach rechts</td></tr>
<tr><td>Bit Or</td><td>|</td><td>links nach rechts</td></tr>
<tr><td>Logical And</td><td>&&</td><td>links nach rechts</td></tr>
<tr><td>Logical Or</td><td>||</td><td>links nach rechts</td></tr>
<tr><td>Conditional</td><td>?:</td><td>rechts nach links</td></tr>
<tr><td>Assignment</td><td>= += -= etc.</td><td>rechts nach links</td></tr>
<tr><td>Expression List</td><td>,</td><td>links nach rechts</td></tr>
</table>

2.5 Kontrollstrukturen

In diesem Abschnitt werden die in der **User Language** verfügbaren Kontrollstrukturen vorgestellt. Kontrollstrukturen definieren die Reihenfolge, in der die Aktionen eines Programms ausgeführt werden. Grundsätzlich wird in der **Bartels User Language** - entsprechend den Grundlagen der strukturierten Programmierung - unterschieden zwischen sequentiellen Programm-Elementen, Alternativen und Repetitionen (englisch: CAR - Concatenation, Alternation, Repetition).

2.5.1 Sequentielle Programm-Elemente

Anweisungen

Eine Anweisung besteht aus einem Ausdruck (siehe Kapitel 2.4) gefolgt von einem Semikolon (`;`). Somit sind z.B.

```
tabulator = '\t' ;
distance = sqrt(a*a+b*b) ;
filename += extension = ".ddb" ;
++ary[i] ;
printf("Part %s ;\n",partname) ;
```

gültige Anweisungen. Fehlt der Ausdruck vor dem Semikolon, hat die Anweisung also die Form

```
;
```

dann liegt liegt der Sonderfall der leeren Anweisung vor. Die leere Anweisung kann in Ausnahmefällen dazu dienen, abhängige Anweisungen (z.B. innerhalb von Schleifen) zu definieren. Grundsätzlich sind sonst aber nur solche Anweisungen sinnvoll, die auch einen Seiteneffekt haben, d.h. irgendeine Variable durch eine Zuweisung verändern oder eine Funktion aktivieren. In der **User Language** sind allerdings auch Anweisungen zulässig, die keine Seiteneffekte haben wie z.B.

```
27+16.3;
++11;
```

Der **User Language Compiler** erkennt Ausdrücke ohne Seiteneffekte und gibt ggf. entsprechende Warnungen aus.

Anweisungen, die in einer Kontext-Abhängigkeit zu einer Alternative bzw. Repetition (siehe unten) stehen, werden im Folgenden als abhängige Anweisungen bezeichnet.

Blöcke

Ein Block besteht aus einer Folge von Vereinbarungen (siehe Kapitel 2.3.2) und Anweisungen und ist in geschweifte Klammern (`{` und `}`) einzuschließen. Ein Block ist syntaktisch äquivalent zur einfachen Anweisung, kann also auch immer anstelle einer einfachen Anweisung angegeben werden.

2.5.2 Alternativen

Alternativen sind Verzweigungen, die die Ausführung bestimmter abhängiger Anweisungen vom Wert eines Ausdrucks abhängig machen.

if- und if-else-Anweisung

Die **`if`**-Anweisung hat die allgemeine Form

```
if (expression)
      statement
```

Die abhängige Anweisung der **`if`**-Anweisung wird nur ausgeführt, wenn der **`if`**-Ausdruck einen vom Nullwert (0 bzw. Leerstring für **`string`**-Ausdruck) verschiedenen Wert besitzt; andernfalls wird sie übersprungen.

Die **`if-else`**-Anweisung besitzt die allgemeine Form

```
if (expression)
      statement
else
      statement
```

Zunächst wird der Ausdruck der **`if-else`**-Anweisung bewertet. Ist dabei das Resultat vom Null-Wert (0 bzw. Leerstring beim **`string`**-Ausdruck) verschieden, wird die erste abhängige Anweisung ausgeführt. Die zweite abhängige Anweisung wird genau dann ausgeführt, wenn das Resultat des Ausdrucks dem Null-Wert entspricht. Die abhängigen Anweisungen der **`if-else`**-Anweisung können selbstverständlich wiederum **`if-else`**-Anweisungen sein. Damit können auch Verschachtelungen und Ketten von **`if-else`**-Anweisungen der Form

```
if (expression)
      statement
else if (expression) {
      if (expression)
              statement
      }
else if (expression)
      statement
:
else
      statement
```

gebildet werden. Bei verschachtelten **`if-else`**-Anweisungen gehört die **`else`**-Anweisung immer zum nächstfrüheren **`if`**, für das noch kein **`else`**-Teil existiert. In folgendem Beispiel wird die Klasse des aktuell geladenen SCM-Elements abgefragt, und der Wert der Variablen **`classname`** wird entsprechend gesetzt:

```
string classname="SCM ";
if (bae_planddbclass()==800)
      classname+="Sheet";
else if (bae_planddbclass()==801 || bae_planddbclass()==803)
      classname+="Symbol/Label";
else if (bae_planddbclass()==802)
      classname+="Marker";
else {
      classname="***INVALID***";
      printf("No valid element loaded!\n");
      }
```

switch Anweisung

Die **switch**-Anweisung sorgt dafür, dass in Abhängigkeit vom Wert eines auf Gleichheit prüfenden Äquivalenz-Ausdrucks bestimmte Anweisungen ausgeführt werden und andere nicht. Die allgemeine Form der **switch**-Anweisung ist gegeben durch

```
switch (expression)
      statement
```

Jeder Anweisung innerhalb der abhängigen **switch**-Anweisung kann eine beliebige Anzahl von **case**-Marken der Form

```
case expression :
```

oder

```
default :
```

vorausgehen. Die zwischen **case**-Marken befindlichen Anweisungen stehen in Abhängigkeit zu den unmittelbar vorhergehenden **case**-Marken. Die abhängigen Anweisungen einer **case**-Marke werden nur dann ausgeführt, wenn der Wert des **switch**-Ausdrucks mit dem Wert des **case**-Ausdrucks übereinstimmt; die **default**-Marke spezifiziert dabei einen beliebigen Wert, d.h. die einer **default**-Marke folgenden abhängigen Anweisungen kommen immer zur Ausführung. **case**-Marken selbst haben keinen Einfluss auf die sequentielle Ausführung einer Reihe von Anweisungen; die Ausführung wird fortgesetzt, als ob keine **case**-Marken vorhanden sind. Um eine **switch**-Anweisung zu verlassen wird üblicherweise in einem **case**-Abschnitt die **break**-Anweisung (siehe auch Kapitel 2.5.4) benutzt. In folgendem Beispiel wird die Klasse des aktuell geladenen SCM-Elements abgefragt, und der Wert der Variablen **classname** wird entsprechend gesetzt:

```
string classname="SCM ";
switch (bae_planddbclass()) {
      case 800 :
      classname+="Sheet";
      break;
      case 801 :
      case 803 :
      classname+="Symbol/Label";
      break;
      case 802 :
      classname+="Marker";
      break;
      default :
      classname="***INVALID***";
      printf("No valid element loaded!\n");
      }
```

2.5.3 Repetitionen

Repetitionen sind Anweisungen, mit deren Hilfe Schleifen zur wiederholten, also repetitiven Abarbeitung bestimmter Teile des Programms definiert werden können. Jede repetitive Anweisung verfügt auch über einen Mechanismus zur Überprüfung einer Ende-Bedingung, bei der die Schleifen-Bearbeitung beendet wird. Wird diese Ende-Bedingung nie erreicht, dann spricht man von einer Endlos-Schleife. Läuft ein Programm während der Abarbeitung in eine derartige Endlos-Schleife, dann kann es den Kontrollfluss nicht mehr von sich aus an den Aufrufer zurückgeben. In solchen Fällen liegt in der Regel ein schwerer Programmierfehler vor. Erkennt der **User Language Compiler** eine Endlos-Schleife, wozu er in bestimmten Fällen in der Lage ist, dann gibt er eine entsprechende Fehlermeldung aus und erzeugt keinen Programm-Code.

while-Anweisung

Die **`while`**-Anweisung hat die allgemeine Form

```
while (expression)
      statement
```

Die abhängige Anweisung wird solange wiederholt, wie der Wert des **`while`**-Ausdrucks ungleich dem Nullwert (0 oder Leerstring für **`string`**-Ausdruck) ist. Das folgende Programm dient dazu, die Inhalte von ASCII-Dateien auf dem Bildschirm anzuzeigen:

```
// ASCII file view
main()
{
      string fname;           // File name
      int fh;                 // File handle
      string curstr="";       // Current input string
      // Set the file error handle mode
      fseterrmode(0);
      // Print the program banner
      printf("ASCII FILE VIEWER STARTED\n");
      // Repeatedly ask for the input file name
      while (fname=askstr("File Name (press RETURN to exit) : ",40)) {
              // Open the input file
              printf("\n");
              if ((fh=fopen(fname,0))==(-1)) {
                      printf("File open failure!\n");
                      continue;
                      }
              // Get the current input string
              while (fgets(curstr,128,fh)==0)
                      // Print the current input string
                      puts(curstr);
              // Test on read errors; close the file
              if (!feof(fh) || fclose(fh))
                      // Read or close error
                      break;
              }
}
```

Die **`continue`**-Anweisung (siehe auch Kapitel 2.5.4) wird in obigem Beispiel verwendet, um wieder an den Beginn der **`while`**-Anweisung zu springen; die **`break`**-Anweisungen (siehe auch Kapitel 2.5.4) dienen dazu, im Fehlerfall die **`while`**-Anweisung zu verlassen.

do-while-Anweisung

Die **`do-while`**-Anweisung hat die allgemeine Form

```
do
      statement
while (expression);
```

Die abhängige Anweisung wird solange wiederholt, bis der Wert des **`do-while`**-Ausdrucks gleich dem Nullwert (0 oder Leerstring für **`string`**-Ausdruck) ist. Die abhängige Anweisung wird also im Unterschied zur **`while`**-Anweisung mindestens einmal ausgeführt.

for-Anweisung

Die **`for`**-Anweisung hat die allgemeine Form

```
for (expression1; expression2; expression3)
      statement
```

und ist äquivalent zu

```
experession1;
while (expression2) {
      statement;
      expression3;
      }
```

Zunächst wird der erste Ausdruck bewertet. Anschließend werden, solange der zweite Ausdruck nicht gleich dem Nullwert (0 bzw. Leerstring für **`string`**-Ausdruck) ist, die abhängige Anweisung ausgeführt und der dritte Ausdruck bewertet. Der erste **`for`**-Ausdruck dient also der Initialisierung, der zweite der Schleifenende-Prüfung, und der dritte typischerweise einer Inkrementierung. Jeder einzelne der drei **`for`**-Ausdrücke darf auch fehlen. Die folgende Funktion **`strwords`** zerlegt einen als Parameter übergebenen **`string`**-Wert in seine einzelnen (durch Blank oder Tabulator) getrennten Worte, speichert diese in einer Liste ab und gibt sie anschließend in umgekehrter Reihenfolge aus:

```
void strwords(string s)
{
      string strlist[];
      int strcount,i,j;
      char c;
      for ( strcount=0,j=0,i=0 ; c=s[i++] ; ) {
              if (c==' ' || c=='\t') {
                      if (j>0) {
                              strcount++;
                              j=0;
                              }
                      continue;
                      }
              strlist[strcount][j++]=c;
              }
      for ( i=strcount ; i>=0 ; i-- )
              printf("%s\n",strlist[i]);
}
```

forall-Anweisung

Die **forall**-Anweisung hat die allgemeine Form

```
forall (identifier1 of identifier2 where expression)
      statement
```

Die **forall**-Anweisung dient der automatischen sequentiellen Abarbeitung der aktuell verfügbaren Elemente eines **index**-Datentyps. Der erste Identifier muss eine **index**-Variable referenzieren, über die der Typ des abzuarbeitenden **forall**-Index spezifiziert wird. Die **forall**-Anweisung sorgt selbsttätig für die Initialisierung und Inkrementierung dieser Variablen. Ist keine Inkrementierung mehr möglich, weil das letzte Element der **index**-Liste erreicht ist, dann ist automatisch auch das Ende-Kriterium erreicht und die Programmausführung wird mit der Anweisung, die der **forall**-Anweisung folgt, fortgesetzt. Die **of**-Anweisung innerhalb der **forall**-Anweisung dient der Spezifikation eines dem **forall**-Index hierarchisch übergeordneten **index**-Wertes, d.h. der zweite Identifier muss eine **index**-Variable referenzieren, innerhalb der die Abarbeitung des **forall**-Index per Definition zulässig ist. Der **where**-Ausdruck schließlich dient dazu, zu entscheiden, ob die abhängige Anweisung für den aktuellen **forall**-Index ausgeführt werden soll (Wert des Ausdrucks ungleich dem Nullwert) oder nicht (Wert des Ausdrucks gleich dem Nullwert). Sowohl die **of**-Anweisung als auch die **where**-Anweisung können wahlweise weggelassen werden, so dass die kürzeste Form der **forall**-Anweisung gegeben ist durch

```
forall (identifier1)
      statement
```

Die **index**-Variablen-Typen sind in Anhang B beschrieben. Folgendes Beispiel gibt für alle platzierten Bauteile der aktuell geladenen physikalischen Netzliste eine Liste der Pins aus, die mit dem Netz **vcc** verbunden sind:

```
index L_CPART part;
index L_CPIN pin;
forall (part where part.USED) {
      forall (pin of part where pin.NET.NAME=="vcc")
              printf("Part %s, Pin %s ;\n",part.NAME,pin.NAME);
```

2.5.4 Kontrollfluss-Steuerung

Neben den bisher vorgestellten Kontrollstrukturen verfügt die **User Language** über einige spezielle Kontrollstrukturen, mit denen der Kontrollfluss zusätzlich gesteuert werden kann.

break-Anweisung

Die **`break`**-Anweisung hat die allgemeine Form

```
break;
```

Die **`break`**-Anweisung muss sich in der Abhängigkeit einer repetitiven (**`while`**, **`do-while`**, **`for`** oder **`forall`**) oder einer **`switch`**-Anweisung befinden (andernfalls Fehlermeldung durch den Compiler). Sie sorgt für den Abbruch der nächstgelegenen repetitiven bzw. **`switch`**-Anweisung, d.h. die Ausführung des Programms wird mit der Anweisung fortgesetzt, die der abgebrochenen Anweisung normalerweise folgt.

continue-Anweisung

Die **`continue`**-Anweisung hat die allgemeine Form

```
continue;
```

Die **`continue`**-Anweisung muss sich in der Abhängigkeit einer repetitiven (**`while`**, **`do-while`**, **`for`** oder **`forall`**) Anweisung befinden (andernfalls Fehlermeldung durch den Compiler). Sie sorgt dafür, dass die Ausführung des Programms an der Stelle fortgesetzt wird, an der über die Wiederholung der nächstgelegenen repetitiven Anweisung entschieden wird.

Funktionsaufruf und return-Anweisung

Sowohl der Funktionsaufruf als auch die **`return`**-Anweisung wurden bereits vorgestellt. Sie seien an dieser Stelle jedoch nochmals aufgeführt, um zu verdeutlichen, dass auch sie den Kontrollfluss entscheidend beeinflussen.

Der Funktionsaufruf ist ein Ausdruck, der einen Sprung zur ersten Anweisung der entsprechenden Funktionsdefinition bewirkt. Die **`return`**-Anweisung darf nur innerhalb von Funktionen verwendet werden. Sie sorgt dafür, dass die Ausführung des Programms unmittelbar nach dem Funktionsaufruf der den Sprung in die Funktion veranlasst hat, fortgesetzt wird, d.h. es erfolgt ein Abbruch der Funktionsausführung, und der Kontrollfluss wird wieder an den Aufrufer der Funktion zurückgegeben. Die allgemeine Form der **`return`**-Anweisung ist gegeben durch

```
return;
```

bzw.

```
return expression;
```

Enthält die **`return`**-Anweisung keinen Ausdruck, dann ist der Rückgabewert des Funktionsaufrufes undefiniert. Im anderen Fall ergibt sich der Rückgabewert der Funktion aus dem **`return`**-Ausdruck, der typ-kompatibel zu dem für die Funktion definierten Datentyp sein muss (ansonsten Fehlermeldung durch den Compiler). Trifft der **User Language Compiler** auf das Ende eines Funktionsblocks, dann erzeugt er, sofern die letzte Anweisung des Blocks nicht eindeutig als **`return`**-Anweisung erkannt werden kann, an dieser Stelle Programm-Code, der einer **`return`**-Anweisung (ggf. mit zum Funktionsdatentyp kompatiblem Null-Wert) entspricht. Die Erzeugung eines Nullwertes ist hierbei allerdings nicht für zusammengesetzte Datentypen möglich. D.h., Funktionsdefinitionen der Form

```
struct structname functionname() { }
```

provozieren eine Compiler-Fehlermeldung, da der **User Language Interpreter** beim lesenden Zugriff auf ein Element des Rückgabewertes einer solchen Funktion in jedem Fall eine Speicherzugriffs-Verletzung (Memory Protection Fault) feststellen würde.

2.6 Preprozessor-Anweisungen

Der **Bartels User Language Compiler** kann spezielle Preprozessor-Anweisungen zur Steuerung der Quelltextabarbeitung verarbeiten. Derartige Anweisungen beginnen mit dem Zeichen `#` und werden durch das Zeilenende begrenzt. Preprozessor-Anweisungen sind an beliebigen Stellen des Quelltextes zulässig; ihr Wirkungsbereich erstreckt sich bis zum Ende des jeweiligen Quelltextes.

2.6.1 Dateieinbindung

Im Sprachumfang der **User Language** ist die aus Standard C bekannte `#include`-Anweisung enthalten. Die `#include`-Anweisung dient dazu, Deklarationen und Funktionen, die in unterschiedlichen Programmen identisch verwendet werden und in einer gesonderten Includedatei abgelegt sind, in die gewünschten Quelldateien einzubinden. Dadurch lässt sich der Aufwand für die Entwicklung und Pflege von **User Language**-Programmen erheblich reduzieren (eine in verschiedenen Programmen benötigte Funktion muss nur einmal im Quelltext definiert werden). Die Syntax für die `#include`-Anweisung lautet:

```
#include "filename" EOLN
```

Die `#include`-Anweisung sorgt dafür, dass der **User Language Compiler** an dieser Stelle mit der Kompilierung der in der `#include`-Anweisung angegebenen Quellcode-Datei beginnt. Diese Anweisung wird durch das Zeilenende abgeschlossen. Ist das Dateiende der Includedatei erreicht, dann wird der Übersetzungsvorgang an der der `#include`-Anweisung nachfolgenden Anweisung fortgesetzt. Innerhalb einer Includedatei sind wiederum `#include`-Anweisungen zulässig, sofern sich dadurch keine Mehrfachbearbeitung identischer Quellcodedateien (Datenrekursion!) ergibt. Bei der Verwendung von `#include`-Anweisungen ist zu berücksichtigen, dass bestimmte Teile der eingebundenen Includedateien in dem jeweiligen Programm u.U. gar nicht benötigt werden. Man sollte beim Compilerlauf daher in jedem Fall den Optimierer aktivieren, um redundante Programmteile zu eliminieren. Das folgende Beispiel zeigt das Include-File **baecall.ulh** mit der Routine `call` zur Aktivierung von **AutoEngineer**-Menüfunktionen:

```
// baecall.ulh -- BAE menu call facilities
void call(int menuitem)       // Call a BAE menu function
{
    // Perform the BAE menu call
    if (bae_callmenu(menuitem))
    {
        // Error; print error message and exit from program
        perror("BAE menu call fault!");
        exit(-1);
    }
}
```

Das folgende Beispiel zeigt den Source-Code für das Programm **ZOOMALL** zur Ausführung des Zoom All-Kommandos im **AutoEngineer**; hierbei wird die über das Include-File **baecall.ulh** eingebundene Routine `call` verwendet:

```
// ZOOMALL -- Call the BAE Zoom All command
#include "baecall.ulh"        // BAE menu call include
main()
{
    // Call Zoom All
    call(101);
}
```

2.6.2 Konstantendefinition

Eine Preprozessor-Anweisung der Form

```
#define IDENT constexpr EOLN
```

sorgt dafür, dass im nachfolgenden Programmtext der über IDENT spezifizierte Identifier jeweils durch den Wert des konstanten Ausdrucks (constexpr) ersetzt wird. Die Anweisung wird durch das Zeilenende abgeschlossen. Diese Anweisung eignet sich insbesondere zur Definition wesentlicher Konstanten.

Die Gültigkeit der Konstantendefinition erstreckt sich bis zum Ende des Programmtextes, sofern die Definition nicht vorher durch eine Preprozessor-Anweisung der Form

```
#undef IDENT EOLN
```

gelöscht wird.

Eine besondere Form der **#define**-Anweisung ist durch die Syntax

```
#define IDENT EOLN
```

gegeben. Hierbei wird lediglich ein Name definiert, dessen Existenz durch die Preprozessor-Anweisungen **#ifdef** bzw. **#ifndef** abgefragt werden kann (siehe hierzu Kapitel 2.6.3).

Typische Beispiele für die Verwendung der **#define**-Anweisung sind:

```
#define DDBCLASSLAYOUT 100
#define mmtoinch 25.4
#define inchtomm 1.0/mmtoinch
#define REPABORT "Operation aborted."
#define ERRCLASS "Operation not allowed for this element!"
#define GERMAN  1
#define ENGLISH 0
#define LANGUAGE GERMAN
#define DEBUG
```

2.6.3 Bedingte Übersetzung

Eine Preprozessor-Anweisung der Form

```
#if constexpr EOLN
```

überprüft, ob der angegebene konstante Ausdruck einen von Null verschiedenen Wert besitzt. Eine Preprozessor-Anweisung der Form

```
#ifdef IDENT EOLN
```

überprüft, ob der über den Identifier spezifizierte Name (durch eine **`#define`**-Anweisung) definiert ist. Eine Preprozessor-Anweisung der Form

```
#ifndef IDENT EOLN
```

überprüft, ob der über den Identifier spezifizierte Name unbekannt ist.

Die auf **`#if`**, **`#ifdef`** bzw. **`#ifndef`** folgenden Zeilen des Programmtextes werden nur dann übersetzt, wenn die entsprechende Bedingung erfüllt. Ansonsten wird der entsprechenden Programmteil lediglich auf korrekte Syntax geprüft.

Auf eine If-Preprozessor-Anweisung kann optional eine Prepozessor-Anweisung der Form

```
#else EOLN
```

folgen. Durch eine Preprozessor-Anweisung der Form

```
#endif EOLN
```

wird die entsprechende If-Konstruktion abgeschlossen. Wenn die If-Bedingung erfüllt ist, dann werden alle Zeilen zwischen **`#else`** und **`#endif`** von der eigentlichen Übersetzung ausgenommen. Ist die If-Bedingung nicht erfüllt, dann wird der Programmteil zwischen der If-Anweisung und **`#else`** (oder in dessen Abwesenheit **`#endif`**) von der Übersetzung ausgenommen. Derartige If-Preprozessor-Konstruktionen dürfen verschachtelt sein.

Das folgende Beispiel illustriert die Möglichkeiten der bedingten Übersetzung:

```
#define ENGLISH 0
#define GERMAN 1
#define LANGUAGE ENGLISH

#if LANGUAGE == GERMAN
#define MSGSTART "Programm gestartet."
#endif
#if LANGUAGE == ENGLISH
#define MSGSTART "Program started."
#endif

#define DEBUG

main()
{
#ifdef DEBUG
    perror(MSGSTART);
#endif
    :
}
```

2.6.4 BNF-Precompiler

In die **Bartels User Language** ist ein BNF-Precompiler zur Realisierung von Interface-Programmen für die Bearbeitung von Fremddatenformaten integriert. Unter Verwendung der zugehörigen Scanner- und Parserfunktionen lassen sich in einfacher und eleganter Weise Programme zur Verarbeitung praktisch beliebiger ASCII-Datenformate implementieren.

Jeder **User Language**-Programmtext darf eine Preprozessor-Anweisung der Form

```
#bnf { ... }
```

enthalten. Das **#bnf**-Statement darf sich über beliebig viele Zeilen des Quelltextes erstrecken. Diese Anweisung aktiviert den BNF-Precompiler des **Bartels User Language Compilers**. BNF (Backus Naur Form) ist ein Formalismus zur Beschreibung der Syntax einer Sprache. Die BNF-Definition (innerhalb der geschweiften Klammern des **#bnf**-Statements) einer Sprache besteht aus einer Folge von Regeln durch die die Grammatik der Sprache (d.h. die in dieser Sprache gültigen Folgen von Worten bzw. Zeichen zur Bildung von Sätzen) festgelegt wird. In der BNF-Notation besteht eine Regel aus einem eindeutigen Grammatikbegriff (non-terminal symbol), dem über den Doppelpunkt-Operator **:** (zu lesen als "besteht aus") eine Folge von einer oder mehreren alternativen (durch das Zeichen **|** voneinander getrennten) Formulierungen zugewiesen wird. Eine Formulierung wiederum besteht aus einer Folge von Grammatikbegriffen und Eingabesymbolen (terminal symbols), wobei auch leere Formulierungen zulässig sind. Der Grammatikbegriff der ersten Regel einer BNF-Definition wird als Startsymbol bezeichnet. Durch die rekursive Verwendung von Grammatikbegriffen innerhalb der Regeln lassen sich unendlich viele bzw. unendlich lange zulässige Sätze definieren. Die Definition einer jeden Regel ist durch einen Strichpunkt **;** abzuschließen.

Die in der BNF-Definition angegebenen Terminalsymbole definieren den Wortschatz, d.h. die Menge der zulässigen Worte der Sprache. Hierbei kennzeichnen die Schlüsselwörter **IDENT** (Identifier), **NUMBER** (numerische Konstante), **SQSTR** (Single Quoted String, Zeichenkette in einfachen Anführungszeichen), **DQSTR** (Double Quoted String, Zeichenkette in Doppelten Anführungszeichen), **EOLN** (End Of Line, Zeilenende **\n**), **EOF** (End Of File; Dateiende bzw. End of String bei der Abarbeitung von Zeichenketten), **EOFINC** (End Of Include File, Ende Dateieinbindung) und **UNKNOWN** (nicht explizit definierte Sonderzeichenketten) nicht näher spezifizierte Terminalsymbole aus den entsprechenden Wortklassen. Die Spezifikation spezieller, anwendungsspezifischer Terminalsymbole geschieht durch die Angabe der entsprechenden Zeichenketten in Anführungszeichen, wobei der BNF-Precompiler automatisch eine Unterteilung in reservierte Worte (Schlüsselwort, Keyword; Identifier bestehend aus mindestens 3 Zeichen) und Operatoren (Single Character Operatoren bestehend aus einem Sonderzeichen bzw. Double Character Operatoren bestehend aus zwei Sonderzeichen) vornimmt.

Beispiele für die Spezifikation von Schlüsselwörtern sind:

```
"SECTION"   'ENDSEC'   'Inch'   'begin'   "end"   'include'
```

Beispiele für die Spezifikation von Double Character Operatoren sind:

```
'<='   '++'   "&&"   "+="   '::'   ':='
```

Beispiele für die Spezifikation von Single Character Operatoren sind:

```
'+'   '-'   "*"   "/"   "="   "["   "}"   '#'   '.'
```

Zwischenräume (Leerzeichen, Tabulatoren, Zeilentrenner) sind üblicherweise bei der Definition einer Grammatik nicht signifikant; sie dienen lediglich der Trennung von benachbarten Symbolen. Auch Kommentare stellen letztendlich nichts anderes als Zwischenräume dar. Von Bedeutung ist in in diesem Zusammenhang lediglich die Definition entsprechender Kommentarbegrenzungs-Operatoren. Per Default trägt der BNF-Precompiler hierfür die Operatoren **/*** (Kommentarstart) und ***/** (Kommentarende) für zeilenübergreifende Kommentare ein. Diese Zuordnung lässt sich mit Hilfe eines speziellen Kommandos am Beginn der BNF-Definition ändern. Die diesbezügliche Anweisung für die Defaulteinstellung lautet wie folgt:

```
COMMENT ('/*', '*/');
```

Durch die Angabe nur eines Kommentarbegrenzungsoperators wie z.B. in

```
COMMENT ('//');
```

werden Kommentare konfiguriert, die sich bis zum Zeilenende erstrecken. Es ist zu beachten, dass die Kommentardefaulteinstellung verloren geht, sobald auch nur eine **COMMENT**-Anweisung in die BNF-Definition eingetragen wird. Will man also z.B. sowohl **/*** und ***/** für zeilenübergreifende Kommentare als auch den Operator **//** für Kommentare bis zum Zeilenende definieren, dann sind explizit die folgenden beiden **COMMENT**-Anweisungen am Beginn der BNF-Definition einzutragen:

```
COMMENT ('/*', '*/');
COMMENT ('//');
```

Eine Besonderheit des **Bartels User Language** BNF-Precompilers besteht darin, dass zu jedem in einer Formulierung aufgeführten Grammatikbegriff bzw. Eingabesymbol optional eine Aktion definiert werden kann. Die Definition einer Aktion besteht aus der in runden Klammern (**(** und **)**) eingeschlossenen Angabe einer **User Language**-Anwenderfunktion, die bei Erkennung des entsprechenden Symbols auszulösen ist. Per Konvention müssen diese Anwenderfunktionen vom Datentyp **int** sein; der Rückgabewert einer solchen Funktion muss Null sein, wenn kein Fehler aufgetreten ist, und (-1) im anderen Fall. Optional ist ein Parameter vom Datentyp **int** zulässig, welcher in der BNF-Definition als Konstante in runden Klammern nach dem Funktionsnamen anzugeben ist.

Die exakte Definition der Syntax der BNF-Notation ist in der Syntaxdefinition der **Bartels User Language** (siehe Kapitel 2.7) enthalten.

Der BNF-Precompiler übersetzt die BNF-Definition in **User Language**-Maschinencode, der eine State-Maschine zur Abarbeitung eines Textes in der definierten Sprache festlegt. Zur Aktivierung dieser State-Maschine stehen die **User Language**-Systemfunktionen **synparsefile** und **synparsestring** zur Verfügung. Die Funktion **synparsefile** aktiviert einen Parser zur Abarbeitung einer durch Dateinamen spezifizierten Eingabedatei während mit der Funktion **synparsestring** eine Zeichenkette anstelle eines Dateiinhaltes abgearbeitet werden kann. Beide Funktionen lösen nach Bedarf automatisch die definierten Aktionen bzw. Anwenderfunktionen aus. Innerhalb dieser Anwenderfunktionen kann mit der Systemfunktion **synscanline** die aktuelle Zeilennummer der Eingabedatei und mit **synscanstring** die aktuell eingelesene Zeichenkette ermittelt werden, wobei die aktuelle Zeichenkette nach Bedarf auch einer semantischen Prüfung unterzogen werden kann. Die Funktionen **synparsefile** bzw. **synparsestring** werden beendet, sobald das Ende der Eingabedatei bzw. des abzuarbeitenden Strings erreicht ist oder ein Syntaxfehler (bzw. ein durch eine Anwenderfunktion erkannter semantischer Fehler) aufgetreten ist.

Der Vollständigkeit halber sei an dieser Stelle noch auf die Systemfunktionen **synscaneoln**, **synscanigncase** und **synparseincfile** hingewiesen. Mit der Scannerfunktion **synscaneoln** kann die Zeilenendeerkennung des BNF-Parsers aktiviert bzw. deaktiviert werden (default: Zeilenendeerkennung deaktiviert). Die Verwendung des Terminalsymbols **EOLN** in der BNF-Definition ist demnach nur sinnvoll bzw. zulässig, wenn für die Dateibearbeitung auch die Zeilenendeerkennung aktiviert wird. Mit der Scannerfunktion **synscanigncase** kann die per Default aktivierte Unterscheidung zwischen Groß- und Kleinschreibung bei der Schlüsselworterkennung deaktiviert bzw. wieder aktiviert werden. Mit der Parserfunktion **synparseincfile** wird die Bearbeitung einer eingebundenen Datei (Includedatei) aktiviert, d.h. der Parser setzt bei Aufruf dieser Funktion den Einlesevorgang unmittelbar am Beginn der namentlich spezifizierten Includedatei fort. Sobald das Ende der Includedatei erreicht ist, wird das **EOFINC** Terminalsymbol als erkannt gemeldet, und der Einlesevorgang wird an der in der übergeordneten Datei unterbrochenen Stelle fortgesetzt. Bei Verwendung der Funktion **synparseincfile** ist demnach auch eine entsprechende Einbindung des Terminalsymbols **EOFINC** in die BNF-Definition erforderlich (andernfalls ist dieses Terminalsymbol obsolet).

Die genauen Beschreibungen der Scanner- und Parserfunktionen finden sich im Anhang C.

Die Funktionalität des BNF-Precompilers soll an dieser Stelle anhand eines Beispiels zur Übernahme von Bauteil-Platzierungsdaten in das aktuell im **Layouteditor** geladene Layout verdeutlicht werden. Die Eingabedaten sollen dabei nach folgendem Schema aufgebaut sein:

```
// This is a comment @

LAYOUT    # This is a comment extending to the end of line

UNITS {
    LENGTH = ( 1.0 INCH ) ;
    ANGLE  = ( 1.0 DEGREE ) ;
    }

PLACEMENT {
    'ic1' : 'dil16' {
        POSITION = (0.000,0.000) ;
        ROTATION = 0.000 ;
        MIRROR   = 0 ;
        }
    'ic2' : 'dil16' {
        POSITION = (2.250,0.100) ;
        }
    'ic3' : 'dil16' {
        POSITION = (1.000,0.394) ;
        ROTATION = 23.500 ;
        }
    'ic4' : 'so16' {
        POSITION = (0.000,0.700) ;
        ROTATION = 0.000 ;
        MIRROR   = 1 ;
        }
    }

END
```

Das Programm zum Einlesen von Platzierungsdaten aus externen Dateien gemäß obigem Beispiel kann unter Verwendung des BNF-Precompilers und der Scanner- und Parserfunktionen auf folgende Weise implementiert werden:

```
// READLPLC -- Read Layout Placement from ASCII File
//_________________________________________________________
// BNF input syntax definition
#bnf {
      COMMENT ("//", "@") ;
      COMMENT ("#") ;
      placefile
              : "LAYOUT" placeunits placedata "END"
              ;
      placeunits
              : "UNITS" "{"
                "LENGTH" "=" "(" floatnum placelengthunit ")" ";"
                "ANGLE" "=" "(" floatnum placeangleunit ")" ";"
                "}"
              ;
      placelengthunit
              : "INCH" (p_unitl(1))
              | "MM" (p_unitl(2))
              | "MIL" (p_unitl(3))
              ;
      placeangleunit
              : "DEGREE" (p_unita(1))
              | "RAD" (p_unita(2))
              ;
      placedata
              : "PLACEMENT" "{" placecommands "}"
              ;
      placecommands
              : placecommands placecommand
              |
              ;
```

```
      placecommand
              : identifier (p_pname) ":" identifier (p_plname)
                "{" placepos placerot placemirror "}" (p_storepart)
              ;
      placepos
              : "POSITION" "="
                "(" floatnum (p_px) "," floatnum (p_py) ")" ";"
              ;
      placerot
              : "ROTATION" "=" floatnum (p_pa) ";"
              |
              ;
      placemirror
              : "MIRROR" "=" NUMBER (p_pm) ";"
              |
              ;
      identifier
              : SQSTR (p_ident)
              ;
      floatnum
              : NUMBER (p_fltnum(0))
              | "-" NUMBER (p_fltnum(1))
              ;
}

//______________________________________________________________
// Globals
double plannx=bae_planwsnx(); // Element origin X coordinate
double planny=bae_planwsny(); // Element origin Y coordinate
double lenconv;               // Length conversion factor
double angconv;               // Angle conversion factor
string curpn;                 // Current part name
string curpln;                // Current physical library name
double curx,cury;             // Current coordinates
double cura = 0.0;            // Current angle (default: 0.0)
int curm = 0;                 // Current mirror flag (default: 0)
string curid;                 // Current identifier
double curflt;                // Current float value
struct partdes {              // Part descriptor
      string pn;              //  Part name
      string pln;             //  Physical library name
      double x,y;             //  Coordinates
      double a;               //  Angle
      int m;                  //  Mirror flag
      } pl[];                 // Part list
int pn=0;                     // Part count

//______________________________________________________________
// Main program
main()
{
      string fname;           // Input file name
      // Test if layout loaded
      if (bae_planddbclass()!=100)
              errormsg("Command not allowed for this element!","");
      // Get and test the placement file name
      if ((fname=askstr("Placement File : ",40))=="")
              errormsg("Operation aborted.","");
      // Parse the placement file
      perror("Reading placement data...");
      parseerr(synparsefile(fname),fname);
      // Perform the placement
      placement();
      // Done
      perror("Operation completed without errors.");
}
```

```
//______________________________________________________________
// Part list management and placement

void gcpart()
// Get or create some part list entry
{
      index L_CPART cpart;    // Part index
      index L_NREF nref;      // Named reference index
      int slb=0;              // Search lower boundary
      int sub=pn-1;           // Search upper boundary
      int idx;                // Search index
      int compres;            // Compare result
      // Loop until search area empty
      while (slb<=sub) {
              // Get the search index
              idx=(slb+sub)>>1;
              // Get and test the compare result
              if ((compres=strcmp(curpn,pl[idx].pn))==0)
                      errormsg("Multiple defined part '%s'!",curpn);
              // Update the search area
              if (compres<0)
                      sub=idx-1;
              else
                      slb=idx+1;
              }
      // Check if part is placed already
      forall (nref where curpn==nref.NAME)
              // Part already placed; abort
              return;
      // Check net list consistence
      forall (cpart where curpn==cpart.NAME) {
              // Check the plname
              if (curpln!=cpart.PLNAME)
                      // Netlist definition mismatch
                      errormsg("Wrong part macro name '%s'!",curpln);
              // Done
              break;
              }
      // Insert the new entry to the part list
      pn++;
      for (idx=pn-2;idx>=slb;idx--)
              pl[idx+1]=pl[idx];
      pl[slb].pn=curpn;
      pl[slb].pln=curpln;
      pl[slb].x=curx;
      pl[slb].y=cury;
      pl[slb].a=cura;
      pl[slb].m=curm;
}

void placement()
// Perform the placement
{
      int i;                  // Loop control variable
      // Iterate part list
      for (i=0;i<pn;i++) {
              // Place the part
              if (ged_storepart(pl[i].pn,pl[i].pln,
               pl[i].x,pl[i].y,pl[i].a,pl[i].m))
                      errormsg("Error placing part '%s'!",pl[i].pn);
              }
}
```

```
//____________________________________________________________
// Error handling

void parseerr(status,fn)
// Handle a syntax/parser error
int status;                  // Scan status
string fn;                   // File name
{
      string msg;            // Error message
      // Evaluate the scan status
      switch (status) {
              case 0 : // No error
              return;
              case 1 :
              msg="No BNF definition available!";
              break;
              case 2 :
              msg="Parser already active!";
              break;
              case 3 :
              sprintf(msg," Error opening file '%s'!",fn);
              break;
              case 4 :
              msg="Too many open files!";
              break;
              case 5 :
              sprintf(msg,"[%s/%d] Fatal read/write error!",
               fn,synscanline());
              break;
              case 6 :
              sprintf(msg,"[%s/%d] Scan item '%s' too long!",
               fn,synscanline(),synscanstring());
              break;
              case 7 :
              sprintf(msg,"[%s/%d] Syntax error at '%s'!",
               fn,synscanline(),synscanstring());
              break;
              case 8 :
              sprintf(msg,"[%s/%d] Unexpected end of file!",
               fn,synscanline());
              break;
              case 9 :
              sprintf(msg,"[%s/%d] Stack overflow (BNF too complex)!",
               fn,synscanline());
              break;
              case 10 :
              sprintf(msg,"[%s/%d] Stack underflow (BNF erroneous)!",
               fn,synscanline());
              break;
              case 11 :
              sprintf(msg,"[%s/%d] Error from parse action function!",
               fn,synscanline());
              break;
              default :
              sprintf(msg,"Unknown parser error code %d!",status);
              break;
              }
      // Print the error message
      errormsg(msg,"");
}

void errormsg(string errfmt,string erritem)
// Print an error message with error item and exit from program
{
      string errmsg;         // Error message string
      // Build and print the error message string
      sprintf(errmsg,errfmt,erritem);
      perror(errmsg);
      // Exit from program
      exit(-1);
}
```

```
//______________________________________________________________
// Parser action routines

int p_unitl(code)
// Handle the length units definition request
// Returns : zero if done or (-1) on error
{
	// Set the length conversion factor
	switch (code) {
		case 1 : lenconv=cvtlength(curflt,1,0); break; // Inch
		case 2 : lenconv=cvtlength(curflt,2,0); break; // mm
		case 3 : lenconv=cvtlength(curflt,3,0); break; // mil
		default : return(-1); // Error
		}
	// Return without errors
	return(0);
}

int p_unita(code)
// Handle the angle units definition request
// Returns : zero if done or (-1) on error
{
	// Set the angle conversion factor
	switch (code) {
		case 1 : angconv=cvtangle(curflt,1,0); break; // Deg
		case 2 : angconv=cvtangle(curflt,2,0); break; // Rad
		default : return(-1); // Error
		}
	// Return without errors
	return(0);
}

int p_storepart()
// Handle the store part request
// Returns : zero if done or (-1) on error
{
	// Get or create the part list entry
	gcpart();
	// Re-init the current angle and mirror mode
	cura=0.0;
	curm=0;
	// Return without errors
	return(0);
}

int p_pname()
// Receive a part name
// Returns : zero if done or (-1) on error
{
	// Store the current part name
	strlower(curpn=curid);
	// Return without errors
	return(0);
}

int p_plname()
// Receive a physical library name
// Returns : zero if done or (-1) on error
{
	// Store the current physical library name
	strlower(curpln=curid);
	// Return without errors
	return(0);
}
```

```
int p_px()
// Receive a part X coordinate
// Returns : zero if done or (-1) on error
{
      // Store the current part X coordinate
      curx=curflt*lenconv+plannx;
      // Return without errors
      return(0);
}

int p_py()
// Receive a part Y coordinate
// Returns : zero if done or (-1) on error
{
      // Store the current part Y coordinate
      cury=curflt*lenconv+planny;
      // Return without errors
      return(0);
}

int p_pa()
// Receive a part angle
// Returns : zero if done or (-1) on error
{
      // Store the current part angle
      cura=curflt*angconv;
      // Return without errors
      return(0);
}

int p_pm()
// Receive a part mirror flag
// Returns : zero if done or (-1) on error
{
      // Get and store the current part mirror flag
      curm=atoi(synscanstring())==0?0:1;
      // Return without errors
      return(0);
}

int p_ident()
// Receive an identifier
// Returns : zero if done or (-1) on error
{
      // Store the current string
      curid=synscanstring();
      // Return without errors
      return(0);
}

int p_fltnum(negflag)
// Receive a float value
// Returns : zero if done or (-1) on error
int negflag;                  // Negative number flag
{
      // Get the current float value
      curflt=atof(synscanstring());
      // Set negative on request
      if (negflag)
              curflt*=(-1);
      // Return without errors
      return(0);
}

//______________________________________________________________
// User Language program end
```

2.6.5 Programmaufruftyp und Undo-Mechanismus

Setzen des Programmaufruftyps

Über die Preprozessoranweisung `#pragma` kann der Aufruftyp des generierten **User Language**-Programms explizit gesetzt werden. Damit kann bei der Kompilierung ungeachtet der Verwendung modulspezifischer Systemfunktionen oder Indexvariablen die Kompatibilität des **User Language**-Programms erweitert oder eingeschränkt werden. Es können die in der nachfolgenden Tabelle angegebenen Aufruftypen spezifiziert werden (siehe hierzu auch Anhang A.1.2).

Aufruftyp	Gültige Interpreterumgebung(en)
`ULCALLERSTD`	alle BAE Programm-Module
`ULCALLERCAP`	alle Schematic Capture Programm-Module
`ULCALLERSCM`	**Schaltplaneditor**
`ULCALLERLAY`	alle Layout Programm-Module
`ULCALLERGED`	**Layouteditor**
`ULCALLERAR`	**Autorouter**
`ULCALLERCAM`	**CAM-Prozessor**
`ULCALLERCV`	**CAM-View**
`ULCALLERICD`	alle IC Design Programm-Module
`ULCALLERCED`	**Chipeditor**

Mit der Preprozessoranweisung

```
#pragma ULCALLERSTD
```

kann der Aufruftyp des generierten **User Language**-Programms explizit auf Standard (STD) gesetzt werden. Ohne obige Preprozessoranweisung quittiert der **User Language Compiler** die Verwendung von Funktionen und Indexvariablen zueinander inkompatibler Aufruftypen mit der Fehlermeldung `Inkompatible Index-/Funktions-Referenz(en)!`. Bei Verwendung obiger `#pragma`-Anweisung ist das generierte Programm in allen Interpreterumgebungen aufrufbar, auch wenn im Programm zur aktuellen Interpreterumgebung inkompatible Indexvariablen oder Funktionen verwendet werden. Es liegt dann in der Verantwortung des Programmierers, den Programmfluss so zu steuern, dass nur solche Indexvariablen bzw. Funktionen tatsächlich referenziert bzw. aufgerufen werden, die zur aktuellen Interpreterumgebung kompatibel sind. Damit lassen sich Programme mit ähnlicher Funktionalität für unterschiedliche Interpreterumgebungen implementieren. Sollte es zur Programmlaufzeit dennoch zu Zugriffen auf inkompatible Indexvariablen oder Funktionen kommen, dann bricht der **User Language Interpreter** das Programm mit einer entsprechenden Fehlermeldung (`UL(Zeile): System-Funktion in dieser Umgebung nicht verfuegbar!`) ab.

Undo-Mechanismus konfigurieren

Die Ausführung eines **User Language**-Programms generiert per Standard einen Undoschritt im BAE-System. Die Präprozessoranweisung

```
#pragma ULCALLERNOUNDO
```

deaktiviert den Undo-Mechanismus für das kompilierte Programm. Damit können redundante Undoschritte für Programme, die keine für das Undo relevanten Operationen ausführen, vermieden werden.

2.7 Syntaxdefinition

Nachfolgend ist die Definition der **User Language**-Syntax in BNF(Backus-Naur-Form)-ähnlicher Notation aufgelistet. Kommentare sind dabei durch **/*** und ***/** begrenzt. Der Doppelpunkt (**:**) entspricht einem Zuweisungs-Operator und ist folglich zu lesen als *"besteht aus"*. Das Zeichen **|** kennzeichnet Alternativen. Identifier werden durch das Symbol **IDENT**, Konstanten durch **NUMBER** (numerisch), **SQSTR** (Zeichen) und **DQSTR** (Zeichenkette) gekennzeichnet. Das Symbol **EOLN** definiert das Zeilenende. Die Terminalzeichen-Sequenzen (also die reservierten Worte und Operatoren) sind fett gedruckt dargestellt.

```
/* User Language Source Code Syntax */

/* Program definition */

program
        : progdefs
        ;

progdefs
        : progdefs progdef
        |
        ;

progdef
        : preproccmd
        | typedef
        | storageclass vardef
        | storageclass fctdef
        ;

/* Preprocessor command */

preproccmd
        : #include DQSTR EOLN
        | #define IDENT optexpr EOLN
        | #undef IDENT EOLN
        | #if expression EOLN
        | #ifdef IDENT EOLN
        | #ifndef IDENT EOLN
        | #else EOLN
        | #endif EOLN
        | #bnf { syntaxdef }
        | #pragma pragmaval
        ;

pragmaval
        : ULCALLERSTD
        | ULCALLERCAP
        | ULCALLERSCM
        | ULCALLERLAY
        | ULCALLERGED
        | ULCALLERAR
        | ULCALLERCAM
        | ULCALLERCV
        | ULCALLERICD
        | ULCALLERCED
        | ULCALLERNOUNDO
        ;

/* Type definition */

typedef
        : typedef declaration
        ;

/* Variable definition */

vardef
        : typespec initdecs ;
        ;
```

```
/* Function definition */

fctdef
        : fcttype IDENT ( fctpars ) fctpardecs { cmditems }         ;

fcttype
        : typespec
        | void
        |
        ;

fctpars
        : fctpardefs
        |
        ;

fctpardefs
        : fctpardefs , IDENT
        | IDENT
        ;

fctpardecs
        : fctpardecs vardef
        |
        ;

/* Storage class */

storageclass
        | static
        | structdef
        ;

/* Type specification */

typespec
        : int
        | double
        | char
        | string
        | index IDENT
        | structdef
        | IDENT
        ;

/* Struct definition */

structdef
        : struct IDENT
        | struct IDENT { members }
        | struct { members }
        ;

members
        : members declaration
        | declaration
        ;

/* Declaration */

declaration
        : typespec decs ;
        ;

decs
        : decs , declarator
        | declarator
        ;

/* Initialized declarator list */
```

```
initdecs
        : initdecs , initdec
        | initdec
        |
        ;

initdec
        : declarator
        | declarator = initializer
        ;

/* Declarator */

declarator
        : IDENT
        | declarator [ ]
        ;

/* Initializer */

initializer
        : assignment
        | { initializers }
        ;

initializers
        : initializers , initializer
        | initializer
        ;

/* Command block */

cmdblock
        : { cmditems }
        | cmditem
        ;

/* Command list */

cmditems
        : cmditems cmditem
        |
        ;

/* Command item */

cmditem
        : preproccmd
        | typedef
        | vardef
        | ifcmd
        | switchcmd
        | forcmd
        | whilecmd
        | docmd
        | forallcmd
        | return optexpr ;
        | break ;
        | continue ;
        | optexpr ;
        ;
```

```
/* If control structure */

ifcmd
        : if ( expression ) cmdblock elsecmd
        ;

elsecmd
        : else cmdblock
        |
        ;

/* Switch control structure */

switchcmd
        : switch ( expression ) { caseblocks }
        ;

caseblocks
        : caseblocks caseblock
        |
        ;

caseblock
        : cases cmditems
        ;

cases
        : cases case
        | case
        ;

case
        : case expression :
        | default :
        ;

/* For control structure */

forcmd
        : for ( optexpr ; optexpr ; optexpr ) cmdblock
        ;

/* While control structure */

whilecmd
        : while ( expression ) cmdblock
        ;

/* Do control structure */

docmd
        : do cmdblock while ( expression ) ;
        ;

/* Forall control structure */

forallcmd
        : forall ( IDENT forallof forallwhere ) cmdblock
        ;

forallof
        : of IDENT
        |
        ;

forallwhere
        : where expression
        |
        ;

/* Expression */
```

```
optexpr
        : expression
        |
        ;

expression
        : expression , assignment
        | assignment
        ;

/* Assignment */

assignment
        : unary = assignment
        | unary |= assignment
        | unary ^= assignment
        | unary &= assignment
        | unary <<= assignment
        | unary >>= assignment
        | unary += assignment
        | unary -= assignment
        | unary *= assignment
        | unary /= assignment
        | unary %= assignment
        | conditional
        ;

/* Conditional evaluation */

conditional
        : log_or
        | log_or ? conditional : conditional
        ;

/* Logical OR */

log_or
        : log_and
        | log_and || log_or
        ;

/* Logical AND */

log_and
        : bit_or
        | bit_or && log_and
        ;

/* Bit OR */

bit_or
        : bit_xor
        | bit_or | bit_xor
        ;

/* Bit Exclusive OR */

bit_xor
        : bit_and
        | bit_xor ^ bit_and
        ;

/* Bit AND */

bit_and
        : equality
        | bit_and & equality
        ;

/* Equivalence comparison */
```

```
equality
        : comparison
        | equality == comparison
        | equality != comparison
        ;

/* Comparison */

comparison
        : shift
        | comparison < shift
        | comparison <= shift
        | comparison > shift
        | comparison >= shift
        ;

/* Shift operations */

shift
        : sum
        | shift << sum
        | shift >> sum
        ;

/* Addition and substraction */

sum
        : product
        | sum + product
        | sum - product
        ;

/* Multiplication and division */

product
        : unary
        | product * unary
        | product / unary
        | product % unary
        ;

/* Unary operators */

unary
        : primary
        | primary ++
        | primary --
        | - unary
        | ! unary
        | ~ unary
        | ++ unary
        | -- unary
        ;

/* Primary operators */

primary
        : IDENT
        | NUMBER
        | SQSTR
        | DQSTR
        | ( expression )
        | IDENT ( optexpr )
        | primary [ expression ]
        | primary . IDENT
        ;
```

```
/* BNF Precompiler syntax definition */

syntaxdef
        : commentdef grammar
        ;

commentdef
        : COMMENT ( commentdel commentend ) ;
        |
        ;

commentend
        : , commentdel
        |
        ;

commentdel
        : SQSTR
        | DQSTR
        ;

grammar
        : grammar rule
        | rule
        ;

rule
        : IDENT : forms ;
        | IDENT : forms | ;
        ;

forms
        : form | forms
        | form
        ;

form
        : form symbol action
        | symbol action
        ;

symbol
        : IDENT
        | SQSTR
        | DQSTR
        | IDENT
        | NUMBER
        | SQSTR
        | DQSTR
        | EOLN
        | EOF
        | EOFINC
        | UNKNOWN
        ;

action
        | ( IDENT ( NUMBER ) )
        | ( IDENT )
        |
        ;

/* BNF syntax description file end */
```

Kapitel 3
Programmiersystem

Dieses Kapitel beschreibt die Bedienung des Programmiersystems der **Bartels User Language**. Dabei werden sowohl die Arbeitsweise des Compilers als auch die Schnittstelle des Interpreters zum **Bartels AutoEngineer** im Detail vorgestellt.

Inhalt

Tabellen

3.1 Konventionen

Das **Bartels User Language**-Programmiersystem besteht aus dem **User Language Compiler** und dem **User Language Interpreter**. Der Compiler hat die Aufgabe, **User Language**-Quellcode in **User Language**-Maschinencode (Programme oder Libraries) zu übersetzen und nach Bedarf statische Linkprozesse durchzuführen bzw. notwendige Informationen für dynamische Linkprozesse zu generieren. Mit Hilfe des Interpreters werden **User Language**-Maschinenprogramme geladen, nach Bedarf dynamisch gelinkt, und schließlich ausgeführt bzw. abgearbeitet. Da **User Language Compiler** und **User Language Interpreter** zeitlich unabhängig voneinander arbeiten, also in unterschiedlichen Programmen implementiert sind, müssen die nachfolgend beschriebenen Konventionen hinsichtlich des Programmzugriffs eingehalten werden.

3.1.1 Programmspeicherung

Der Compiler legt fehlerfrei übersetzte Programme und Libraries unter ihrem jeweiligen Namen in der Datei `ulcprog.vdb` im Programmverzeichnis des **Bartels AutoEngineer** ab. Mit dem Maschinencode wird dabei die Versionsnummer des **User Language Compilers** sowie der Aufruftyp (eine Kodierung für die zum Programm kompatible Interpreterumgebung) abgelegt. Darüber hinaus werden nach Bedarf auch notwendige Informationen für dynamische (d.h. zur Laufzeit durchzuführende) Linkprozesse abgespeichert. Der Interpreter lädt beim Programmaufruf das entsprechende Programm über seinen Namen aus der Datei `ulcprog.vdb` des BAE-Programmverzeichnisses. Dabei wird mit Hilfe der **User Language**-Versionsnummer geprüft, ob das Programm mit einer zum Interpreter kompatiblen Compiler-Version erzeugt wurde (andernfalls bestünde die Gefahr, dass der Interpreter die Programmstruktur gar nicht versteht). Über den Aufruftyp erfolgt die Prüfung der Kompatibilität des Programms zur aktuellen Interpreterumgebung; dies ist notwendig, um sicherzustellen, dass während der Programmausführung keine Index-Variablen-Typen oder Systemfunktionen referenziert werden, die in der aktuellen Interpreterumgebung gar nicht implementiert sind. Während der Programmladephase führt der Interpreter nach Bedarf automatisch die für den Programmlauf notwendigen dynamischen Linkprozesse durch, wobei die einzubindenden Libraries ebenfalls einer Kompatibilitätsprüfung unterzogen werden.

3.1.2 Maschinenarchitektur

Die in der **User Language** implementierte Maschinenarchitektur entspricht der einer Stackmaschine. Der darin definierte Befehlssatz (Instruction Set) umfasst Ladebefehle (zum Laden von Variablen- oder Konstantenwerten), ALU-Befehle (zur Aktivierung der arithmetisch-logischen Einheit der Maschine), Speicherbefehle (für Zuweisungen), Sprungbefehle (zur Realisierung von Alternativen und Repetitionen), Befehle zum Aufruf und Beendigen von Funktionen, sowie Befehle zur direkten Manipulation des Stacks.

Der Befehlssatz (Instruction Set) dieser Maschine ist in Tabelle 3-1 aufgelistet; die in den Stackspalten eingetragenen Werte geben dabei für jede Instruktion an, wie viele Argumente auf dem Stack erforderlich sind, und wie die Stackgröße durch die Abarbeitung der Instruktion verändert wird.

Tabelle 3-1: User Language Maschinen-Befehlssatz

Instruktion	Stack-argumente	Stack-änderung	Instruktionsbezeichnung
nop	0	0	No operation
add	2	-1	Add
addstr	2	-1	Add string
and	2	-1	And
bnot	1	0	Binary not
cmpeq	2	-1	Compare equal
cmpge	2	-1	Compare greater equal
cmpgt	2	-1	Compare greater
cmple	2	-1	Compare less equal
cmplt	2	-1	Compare less
cmpne	2	-1	Compare not equal
decr	1	0	Decrement
div	2	-1	Divide
divr	2	-1	Divide rest
incr	1	0	Increment
mul	2	-1	Multiply
neg	1	0	Negate
not	1	0	Not
or	2	-1	Or
shl	2	-1	Shift left
shr	2	-1	Shift right
sub	2	-1	Subtract
xor	2	-1	Exclusive or
cast t	1	0	Cast value
castoiv i	3	-2	Cast of index variable
getary	2	-1	Get array element
getidx i	1	1	Get index
getidxof i	3	-1	Get index of
loadas s	1	0	Load stack array element
loadav v	1	0	Load variable array element
loadchr c	0	1	Load character

Instruktion	Stack-argumente	Stack-änderung	Instruktionsbezeichnung
loaddbl d	0	1	Load double
loadint i	0	1	Load integer
loadiv v	2	-1	Load index variable
loadoiv v	4	-3	Load of index variable
loads s	0	1	Load stack
loadsd s	0	1	Load stack destructive
loadstr s	0	1	Load string
loaduref f	0	1	Load user function reference
loadv v	0	1	Load variable
loadvd v	0	1	Load variable destructive
storeas s	2	-2	Store stack array element
storeav v	2	-2	Store variable array element
stores s	1	-1	Store stack
storev v	1	-1	Store variable
pop s	0	0	Pop stack
popt	1	-1	Pop top of stack
push s	0	0	Push stack
swap s	0	0	Swap stack
xchg s	0	0	Exchange stack
xchgt	2	0	Exchange top of stack
jump p	0	0	Jump always
jumpeq p	2	-1	Jump if stack tops equal
jumpnz p	1	-1	Jump if stack nonzero
jumpz p	1	-1	Jump if stack zero
calls f	0	1	Call system function
callu f	0	1	Call user function
hlt	0	0	Halt program
ret	1	-1	Return (pop optional stack)
stop	0	0	Stop function

3.2 Compiler

Der **User Language Compiler** dient dazu, **User Language**-Quelltext in **User Language**-Maschinenprogramme bzw. in **User Language**-Libraries zu übersetzen. **User Language**-Programme können vom **User Language Interpreter** ausgeführt werden. **User Language**-Libraries werden üblicherweise aus häufig benötigten Quelltexten erzeugt. Der Maschinencode von **User Language**-Libraries kann wahlweise statisch (während der Kompilierung durch den **User Language Compiler**) oder dynamisch (während der Laufzeit durch den **User Language Interpreter**) eingebunden werden in anderen Maschinencode (Programme oder Libraries). Der Vorteil des Librarykonzepts besteht darin, dass häufig benötigte Quelltexte *nur einmal* der zeitaufwändigen Kompilierung unterzogen werden müssen; anschließend kann der entsprechende Maschinencode über die sehr viel schnelleren Linkprozesse referenziert werden.

3.2.1 Arbeitsweise

Der **User Language Compiler** übersetzt **User Language**-Quelltext in **User Language**-Maschinencode (**User Language**-Programme oder **User Language**-Libraries). Bei der Kompilierung führt der Compiler eine umfassende Konsistenzprüfung des Quelltextes hinsichtlich Syntax und Semantik durch, befreit das Programm bzw. die Library von Redundanzen und erzeugt schließlich - in sehr kompakter Form - einen zum Quelltext äquivalenten Maschinencode. Nach Bedarf werden durch den im **User Language Compiler** integrierten Linker statische Linkprozesse durchgeführt bzw. notwendige Informationen für dynamische Linkprozesse generiert. Dies alles geschieht in der nachfolgend beschriebenen Abfolge von Arbeitsschritten.

Syntaxanalyse und Semantikprüfung

Im ersten Schritt der Kompilierung erfolgt die syntaktische Analyse des Quelltextes. Über einen Parser-Lauf werden dabei die formalen Probleme behandelt, ob also die in der Quelltextdatei enthaltene Sequenz von Worten und Zeichen ein syntaktisch gültiges **User Language**-Programm (bzw. eine korrekte **User Language**-Library) darstellt. Bereits während der Syntaxanalyse wird der Teil der Semantikprüfung durchgeführt, der die im Programm enthaltenen Definitionen (für Variablen, Parameter und Funktionen) auf Konsistenz und Eindeutigkeit in Bezug auf deren Geltungsbereiche prüft. Ergebnis dieses ersten Parser-Laufes (Pass 1) ist eine interne Symboltabelle, die für den zweiten Parser-Lauf (Pass 2) zur semantischen Prüfung benötigt wird. Die semantische Prüfung umfasst die kontext-sensitive Analyse des Quelltextes. Ziel dabei ist es, den Missbrauch der im Programm bzw. in der Library definierten Objekte zu unterbinden. Es wird also die Gültigkeit der Verwendung von Namen ebenso geprüft, wie die Zulässigkeit der Operationen auf die im Programm bzw. in der Library definierten Objekte.

Generierung des Maschinencodes

Bereits während der Semantikprüfung, d.h. im zweiten Parser-Durchlauf (Pass 2) wird der zum Quelltext äquivalente Maschinencode konstruiert. Nur wenn die semantische Prüfung vollständig durchgeführt wurde, und dabei keine Fehler auftraten, entspricht der erzeugte Maschinencode auch einem ausführbaren Maschinenprogramm bzw. einer korrekten **User Language**-Library.

Linker

Mit dem im **User Language Compiler** integrierten Linker können optional statische Linkprozesse durchgeführt bzw. notwendige Informationen zur Durchführung dynamischer Linkprozesse generiert werden.

Beim statischen Linken (Compiler-Option **`-lib`**) bindet der **User Language Compiler** den Maschinencode der angeforderten **User Language**-Libraries direkt in den Maschinencode des aktuell übersetzten Quelltextes ein. Dabei werden die angeforderten Libraries sowie die darin referenzierten Objekte (Funktionen, Variablen, etc.) einer Kompatibilitäts- und Konsistenzprüfung unterzogen.

Beim dynamischen Linken (Compiler-Option **`-dll`**) simuliert der **User Language Compiler** lediglich die Einbindung des Maschinencodes der angeforderten **User Language**-Libraries. Resultat dieses Prozesses sind Relokationstabellen mit Informationen zur Auflösung der Bibliotheksreferenzen. Diese Relokationstabellen werden mit dem zu übersetzenden Maschinencode abgespeichert und später vom **User Language Interpreter** zur Laufzeiteinbindung der angeforderten Dynamic Link Libraries verwendet. Bei Änderungen an **User Language**-Libraries ist zu beachten, dass alle diejenigen **User Language**-Programme und **User Language**-Libraries, die Anforderungen zum dynamischen Linken der geänderten Libraries enthalten, neu kompiliert werden müssen.

Optimierer

Der Optimierer des **User Language Compilers** kann mit der Option `-O` aktiviert werden. Er hat die Aufgabe, den zuvor erzeugten Maschinencode von Redundanzen zu befreien und den Maschinencode durch Änderungen effizienter zu gestalten. Der Optimierer erkennt und eliminiert dabei nicht erreichbare Code-Segmente sowie nicht referenzierte Funktions-, Variablen- und Parameterdefinitionen. Er wandelt soweit möglich Variablenreferenzen in Konstantenzugriffe um (Constant Propagation), und er führt algebraische Optimierungen durch. Durch den Einsatz des Optimierers ergibt sich in aller Regel eine signifikante Reduzierung des Speicherplatzbedarfs und der Laufzeit für den Maschinencode ergibt.

Auf einen ausgesprochen nützlichen Nebeneffekt der Optimierung sei ausdrücklich hingewiesen: in einem durch den Optimierer modifizierten Maschinencode lassen sich u.U. Programmierfehler lokalisieren, die der Compiler sonst nicht erkannt hätte.

Prüfung des Maschinencodes

Nachdem der Maschinencode fertig erzeugt ist, wird er nochmals auf schwerwiegende Programmierfehler geprüft. Zu dieser Kategorie von Fehlern zählen die Division durch Null, die Konstruktion von Endlos-Schleifen und endlos-rekursive Funktionsaufrufe, die der Compiler durch die Analyse des Maschinencodes u.U. erkennen kann.

Ausgabe einer Listingdatei

Der **User Language Compiler** kann optional zur Ausgabe einer Listingdatei veranlasst werden. Die Angaben in dieser Datei sind u.U. nützlich bei der Lokalisierung von Fehlern, die erst zur Laufzeit, also bei der Programmausführung auftreten. Der vollständige Inhalt der Listingdatei besteht aus allgemeinen Angaben zum Programm bzw. zur Library (Name, Version, Aufruftyp), Verweisen auf die im Maschinencode referenzierten Libraries bzw. Library-Definitionen, der Auflistung der im Maschinencode statisch eingebundenen Libraries und der im Maschinencode definierten Objekte (Funktionen, Variablen, Strukturen, usw.) sowie der Auflistung des Maschinencodes selbst (Liste der Instruktionen mit Angabe der Quelltextzeilennummer und der äquivalenten Zeilennummer(n) im Maschinencode).

Die Ausführlichkeit der auszugebenden Information kann über verschiedene Modi der Listing-Option `-l` des **User Language Compilers** kontrolliert werden, wodurch z.B. vom Inhalt her eingeschränkte Listingdateien zur Dokumentation von **User Language**-Libraries (Funktionsreferenzen) erzeugt werden können.

Die Option `-ld` gestattet die Spezifikation eines alternativen Verzeichnisses für die mit der Option `-l` auszugebenden Listindateien. Dies ist insbesondere bei der Verwendung von `make`-Utilities zur automatisierten Compilierung geänderter **User Language**-Programme nützlich, um das Quellverzeichnis "sauber" zu halten. Im Verzeichnis `baeulc` wird ein `makefile` bereitgestellt, das die Abhägigkeiten der **User Language**-Programme von Includedateien enthält und mit Listingdateien in einem Unterverzeichnis (`lst`) arbeitet.

Speicherung des Maschinencodes

Der letzte Schritt des Compiler-Laufes besteht in der Speicherung des erzeugten Maschinenprogramms bzw. der generierten **User Language**-Library. Der **User Language Compiler** legt per Default das Maschinenprogramm (bzw. die **User Language**-Library) als Datenbankeintrag unter dem beim Compileraufruf spezifizierten Elementnamen (siehe Compiler-Optionen `-Source`, `-cp`, `-cl`) in der Datei `ulcprog.vdb` im Programmverzeichnis des **Bartels AutoEngineer** ab. **User Language**-Programme und **User Language**-Libraries werden dabei jeweils einer eigenen Datenbankklasse zugeordnet.

Mit Hilfe spezieller Compiler-Optionen ist auch das Löschen von **User Language**-Programmen (Option `-dp`) und **User Language**-Libraries (Option `-dl`) aus der Datei `ulcprog.vdb` möglich. Damit kann die Datei `ulcprog.vdb` von veraltetem bzw. nicht mehr benötigtem Maschinencode bereinigt werden.

3.2.2 Compileraufruf

Der Compileraufruf startet die Übersetzung von **User Language**-Programmen bzw. von **User Language**-Libraries.

Synopsis

Der Aufruf des **User Language Compilers** erfolgt auf Betriebssystemebene. Die Aufrufsyntax lautet:

```
ulc [-wcon|-wcoff] [[-S[ource]] srcfile...]
    [-lib libname...] [-dll libname...]
    [{-cp|-cl} [dstname...]]
    [-I[nclude] includepath...] [-D[efine] macroid...]
    [-O[0|1]] [-e[0|1]] [-w[0|1|2|3|4]] [-t[0|1]]
    [-l[0|1|2|3|4|5]] [-ld listingdirectory]
    [-dp prgname...] [-dl libname...]
    [-ulp prgfilename] [-ull libfilename]
    [-log logfilename]
```

Bei einem syntaktisch falschen Aufruf des **User Language Compiler** wird die korrekte Aufrufsyntax des Compilers angezeigt, und der Compiler-Lauf wird abgebrochen.

Optionen

Die Kommandozeilenoptionen des **User Language Compilers** bestehen aus einem Bindestrich (**-**) oder einem Schrägstrich (**/**) gefolgt von der Optionsspezifikation. Optionspezifikationen, die nur aus einem Buchstaben und der wahlweisen Angabe einer numerischen Modus- oder Schalterangabe bestehen, bezeichnet man häufig als Switches oder Flags. Diese speziellen Optionen können wahlweise gruppiert werden wie z.B. in **/l2Ow3** oder **-O1w3l2**, wo jeweils der Modus 2 für die Listingausgabe selektiert, der Optimierer aktiviert und der Warning Severity Level auf 3 gesetzt werden.

Wildcard Option [-wcon|-wcoff]

Mit Hilfe dieser Option kann die Berücksichtigung von Wildcards bei der Spezifikation von Datei- und Elementnamen aktiviert (Option **-wcon**; Default) bzw. deaktiviert (Option **-wcoff**) werden. Ist die Wildcarderkennung aktiviert, dann erlangen die Zeichen **?** und ***** Sonderbedeutung bei der Spezifikation von Datei- und Elementnamen; **?** ist dann Platzhalter für ein beliebiges Zeichen, ***** ist dann Platzhalter für eine beliebige Anzahl beliebiger Zeichen. Die Deaktivierung der Wildcarderkennung ist notwendig, um die Bearbeitung von Programmnamen wie z.B. **scm_?** oder **ged_*** zu ermöglichen.

Source File Option [[-S[ource]] srcfile...]

Mit dieser Option werden die Namen der zu übersetzenden Quelltextdateien spezifiziert. Die Dateinamen dürfen dabei Verzeichnispfade enthalten, d.h. die Quelltextdateien müssen nicht notgedrungen im aktuellen Verzeichnis abgelegt sein. Bei der Auswertung von Dateinamen werden Wildcards berücksichtigt, sofern die Wildcarderkennung mit der Option **-wcon** (siehe oben) aktiviert ist. Dateinamen können wahlweise mit oder ohne Namenserweiterung spezifiziert werden. Wird die Namenserweiterung weggelassen, dann für der Compiler automatisch die Extension **.ulc** an die entsprechenden Dateinamen an. Quelltextdateinamen mit anderen Namenserweiterungen der müssen also mit ihrer Extension spezifiziert werden. Damit ist es dann allerdings auch möglich, z.B. **User Language**-Libraries aus Includedateien mit der Extension **.ulh** zu erzeugen. Der Type des zu erzeugenden Maschinencodes wird mit den Optionen **-cp** (für **User Language**-Programme; siehe unten) bzw. **-cl** (für **User Language**-Libraries) festgelegt. Der Elementname des erzeugten Maschinencodes ergibt sich aus dem Quelltextdateinamen, wobei der Verzeichnispfad und die Dateinamenserweiterung weggelassen werden. Ein von dieser Konvention abweichender Name für den Maschinencode kann mit den Optionen **-cp** bzw. **-cl** (siehe unten) spezifiziert werden. Das Schlüsselwort **-Source** bzw. **-S** ist bei der Spezifikation von Quelltextendateinamen nur dann erforderlich, wenn sich sonst Mehrdeutigkeiten bei der Auswertung der Kommandozeile ergeben könnten. Dies ist z.B. dann nicht der Fall, wenn die Quelltextdateinamen die ersten Namensargumente im Compileraufruf darstellen. Mit Hilfe der Schlüsselworte **-Source** bzw. **-S** können andererseits jedoch an beliebigen Stellen in der Kommandozeile Quelltextdateien spezifiziert. Ist weder die Option **-dp** noch die Option **-dl** (siehe unten) spezifiziert, dann erwartet der **User Language Compiler** die Angabe von zumindest einer Quelltextdatei.

Static Link Option [-lib libname...]

Die Static Link Option **-lib** erwartet den Namen einer oder mehrerer **User Language**-Libraries sowie die Spezifikation von zumindest einer Quellcodedatei (siehe oben, Option **-Source**). Die mit **-lib** angeforderten Libraries müssen in kompilierter Form in der Datei **ulcprog.vdb** im BAE-Programmverzeichnis verfügbar sein. Der im **User Language Compiler** integrierte Linker bindet den Maschinencode der angeforderten Libraries in den Maschinencode der zu übersetzenden Quelltextdateien ein.

Dynamic Link Option [-dll libname...]

Die Dynamic Link Option **-lib** erwartet den Namen einer oder mehrerer **User Language**-Libraries sowie die Spezifikation von zumindest einer Quellcodedatei (siehe oben, Option **-Source**). Die mit **-dll** angeforderten Libraries müssen in kompilierter Form in der Datei **ulcprog.vdb** im BAE-Programmverzeichnis verfügbar sein. Der im **User Language Compiler** integrierte Linker erzeugt die für den **User Language Interpreter** notwendigen Informationen zur Laufzeiteinbindung der angeforderten Libraries in den Maschinencode der zu übersetzenden Quelltextdateien.

Create Program/Library Option [{-cp|-cl} [dstname...]]

Mit dieser Option wird der Typ des zu erzeugenden Maschinencodes festgelegt. Der **User Language Compiler** kann sowohl **User Language**-Programme als auch **User Language**-Libraries generieren. Per standard, also wenn weder die Option **-cp** noch die Option **-cl** ist, erzeugt der Compiler **User Language**-Programme. Mit der Option **-cp** kann die Generierung von Programmen explizit veranlasst werden, die Option **-cl** hingegen veranlasst die Generierung von Libraries; es dürfen nicht beide Optionen gleichzeitig angegeben werden. Der Elementname des zu erzeugenden Maschinencodes ergibt sich standardmäßig aus dem jeweiligen Quellcodedateinamen durch Elimination des Verzeichnispfades und der Namenserweiterung. Abweichend von dieser Konvention können mit den Optionen **-cp** und **-cl** explizit andere Elementnamen für zu erzeugenden Maschinencode angegeben werden, wobei dann aber nur noch *genau eine* Quelltextdatei spezifiziert werden darf (siehe oben, Option **-Source**). Der generierte Maschinencode wird unter dem spezifizierten Zielelementnamen in der Datei **ulcprog.vdb** im BAE-Programmverzeichnis abgelegt. Wildcards werden bei der Angabe von Zielelementnamen für Maschinencode grundsätzlich nicht berücksichtigt. Die explizite Angabe mehrerer Zielelementnamen dagegen ist zulässig; damit ist es möglich den Maschinencode einer einzelnen Quelltextdatei unter verschiedenen Namen abzulegen, also z.B. aus der Quelltextdatei **bae_st.ulh** mit einem einzigen Compileraufruf die Programme **scm_st**, **ged_st**, usw. zu erzeugen.

Include Path Option [-I[nclude] includepath...]

Mit der Option **-Include** (bzw. **-I**) können Alternativpfade für die Suche nach Includedateien spezifiziert werden. Diese Option erwartet zumindest einen Verzeichnispfadnamen als Argument. Stößt der Compiler im Quelltext auf eine **#include**-Anweisung, dann sucht er zunächst im aktuellen Verzeichnis nach der angeforderten Includedatei und dehnt die Suche anschließend auf die mit der Option **-Include** angegebenen Verzeichnisse aus, wobei die Verzeichnisse in der Reihenfolge ihrer Spezifikation abgesucht werden.

Define Option [-D[efine] macroid...]

Mit der Option **-Define** (bzw. **-D**) können beim Compileraufruf Makros definiert werden. Diese Option erwartet zumindest einen Makronamen als Argument. Die Option **-Define** entspricht der **#define**-Anweisung im Quelltext, d.h. die mit dieser Option definierten Makros können in den Preprozessoranweisungen **#ifdef** und **#ifndef** zur Steuerung der bedingten Übersetzung ausgewertet werden.

Optimizer Option [-O[0|1]]

Mit der Option **-O** kann der Optimierer des **User Language Compilers** aktiviert bzw. deaktiviert werden. Per Default (d.h., wenn diese Option nicht spezifiziert ist) ist der Optimierer deaktiviert. Die Option **-O** bzw. **-O1** aktiviert den Optimierer. Mit der Option **-O0** kann der Optimierer explizit deaktiviert werden. Der Optimierer befreit den Maschinencode von Redundanzen, und gestaltet ihn durch Modifikationen effizienter. Optimierter Maschinencode benötigt in der Regel erheblich weniger Festplatten- und Hauptspeicher und kann schneller geladen und abgearbeitet werden. Es wird daher dringend empfohlen, den Optimierer zu aktivieren.

Error Severity Option [-e[0|1]]

Mit der Option **-e** kann der Error Severity Level gesetzt werden. Per default (d.h., wenn diese Option nicht spezifiziert ist) ist der Wert 1 eingestellt. Die Option **-e0** setzt den Error Severity Level auf 0; die Option **-e** bzw. **-e1** setzt den Error Severity Level explizit auf 1. Ist der Error Severity Level auf 1 eingestellt, dann versucht der Compiler *alle* beim Compileraufruf spezifizierten Quelltextdateien zu übersetzen (ungeachtet etwaiger Fehler beim übersetzen einzelner Quelltexte); ein Error Severity Level von 0 hingegen veranlasst den Compiler, den Übersetzungsvorgang bei der ersten fehlerhaften Quelltextdatei abzubrechen.

Warning Severity Option [-w[0|1|2|3|4]]

Mit der Option **-w** kann der Warning Severity Level auf einen Wert von 0 bis 4 gesetzt werden. Per Default (d.h., wenn diese Option nicht spezifiziert ist) ist der Wert 0 eingestellt. Wird diese Option ohne die explizite Angabe eines Levels angegeben (**-w**), dann wird der Wert 3 eingestellt. Jede im Compiler definierte Warnmeldung ist einem speziellen Warning Severity Level zugeordnet, wobei höhere Werte unwichtigere Warnungen kennzeichnen. Der Compiler gibt nur die Warnungen aus, deren Warning Severity Level kleiner oder gleich dem mit der Option **-w** eingestellten Level ist, d.h. mit dieser Option kann die Ausgabe weniger bedeutsamer Warnmeldungen unterdrückt werden.

Top Level Warnings Only Option [-t[0|1]]

Mit der Option **-t** können wahlweise Warnungen mit Bezug auf die Kompilierung von Includedateien unterdrückt werden (Wert 1). Per Default (d.h., wenn diese Option nicht spezifiziert ist bzw. der Optionswert 0 gesetzt ist) werden Warnmeldungen mit Bezug auf alle kompilierten Quellcodedateien ausgegeben. Ist dieser Optionswert auf 1 gesetzt, dann unterdrückt der **User Language Compiler** die Ausgabe von Warnmeldungen mit Bezug auf die Kompilierung von Includedateien und gibt lediglich die Warnmeldungen mit Bezug auf die "Top-Level"-Quellcodedatei(en) aus. Dies reduziert die Anzahl der Warnmeldungen und erleichtert damit deren Analyse insbesondere dann, wenn mit Standardincludedateien gearbeitet wird, die (eine Vielzahl von) Funktionen und Variablen enthalten, welche nicht in jedem Programm verwendet werden.

Listing Option [-l[0|1|2|3|4|5]]

Mit der Option **-l** kann die Listingausgabe gesteuert werden. Dabei können Listingmode im Bereich von 0 bis 5 spezifiziert werden. Mit dem Listingmodus 0 wird keine Listingausgabe erzeugt, während der Modus 5 die detailliertesten Informationen liefert. Modus 0 ist der Standardwert, d.h. wenn die Option **-l** nicht angegeben ist, dann erfolgt auch keine Listingausgabe. Wird diese Option ohne die explizite Angabe des Modus spezifiziert (**-l**), dann wird der Modus 5 eingestellt. Der Name der Listingdatei aus dem Namen der Quelltextdatei durch Abändern der Dateinamenserweiterung in **.lst** erzeugt. Die Listingdatei wird vom System nicht weiter benötigt, sie ist lediglich zur Auswertung durch den Benutzer bestimmt.

Listing Directory Option [-ld listingdirectory]

Die Option **-ld** gestattet die Spezifikation eines alternativen Verzeichnisses für die mit der Option **-l** auszugebenden Listindateien. Dies ist insbesondere bei der Verwendung von **make**-Utilities zur automatisierten Compilierung geänderter **User Language**-Programme nützlich, um das Quellverzeichnis "sauber" zu halten. Im Verzeichnis **baeulc** wird ein **makefile** bereitgestellt, das die Abhägigkeiten der **User Language**-Programme von Includedateien enthält und mit Listingdateien in einem Unterverzeichnis (**lst**) arbeitet.

Delete Program Option [-dp prgname...]

Mit der Option **-dp** können einmal übersetzte **User Language**-Programme wieder aus der Datei **ulcprog.vdb** im BAE-Programmverzeichnis gelöscht werden. Diese Option erwartet den Elementnamen zumindest eines Programms als Argument. Bei der Auswertung der Elementnamen werden Wildcards berücksichtigt, sofern die Wildcarderkennung mit der Option **-wcon** (siehe oben) aktiviert ist. Beim Versuch, nicht-existente Libraries zu löschen, werden Warnungen ausgegeben. Damit nicht Maschinencode gelöscht werden kann, der unmittelbar zuvor im gleichen Compiler-Lauf erzeugt wurde, erfolgt das Löschen von Programmen grundsätzlich immer *bevor* mit der Übersetzung von Quelltexten begonnen wird.

Delete Library Option [-dl libname...]

Mit der Option `-dl` können einmal übersetzte **User Language**-Libraries wieder aus der Datei `ulcprog.vdb` im BAE-Programmverzeichnis gelöscht werden. Diese Option erwartet den Elementnamen zumindest einer Library als Argument. Bei der Auswertung der Elementnamen werden Wildcards berücksichtigt, sofern die Wildcarderkennung mit der Option `-wcon` (siehe oben) aktiviert ist. Beim Versuch, nicht-existente Libraries zu löschen, werden Warnungen ausgegeben. Damit nicht Maschinencode gelöscht werden kann, der unmittelbar zuvor im gleichen Compiler-Lauf erzeugt wurde, erfolgt das Löschen von Libraries grundsätzlich immer *bevor* mit der Übersetzung von Quelltexten begonnen wird.

Program Database File Name Option [-ulp prgfilename]

Per Default speichert der **User Language Compiler User Language**-Programme in der Datei `ulcprog.vdb` im Programmverzeichnis des **Bartels AutoEngineer** ab. Mit der Option `-ulp` kann hierfür eine andere Datei angegeben werden.

Library Database File Name Option [-ull libfilename]

Per Default speichert der **User Language Compiler User Language**-Libraries in der Datei `ulcprog.vdb` im Programmverzeichnis des **Bartels AutoEngineer** ab. Mit der Option `-ull` kann hierfür eine andere Datei angegeben werden.

Log File Option [-log logfilename]

Der **User Language Compiler** gibt alle Meldungen auf die Standardausgabe und gleichzeitig auf eine Reportdatei aus. Die Ausgabe auf die Reportdatei erfolgt, um längere Listen von Meldungen, welche insbesondere bei der Übersetzung mehrerer Quelltextdateien entstehen können, zu sichern. Der Standardname der Reportdatei ist `ulc.log` (im aktuellen Verzeichnis). Mit der Option `-log` kann ein anderer Dateiname für die Reportdatei angegeben werden.

Beispiele

Kompilieren des in **ulprog.ulc** enthaltenen **User Language**-Programms mit Optimierung und Ausgabe von Warnmeldungen; das übersetzte Programm wird unter dem Namen **ulprog** in der Datei **ulcprog.vdb** im BAE-Programmverzeichnis abgelegt:

```
ulc ulprog -Ow
```

Kompilieren des in **ulprog.ulc** enthaltenen **User Language**-Programms mit Erzeugung einer Listingdatei (**ulprog.lst**); das übersetzte Programm wird unter dem Namen **newprog** in der Datei **ulcprog.vdb** im BAE-Programmverzeichnis abgelegt:

```
ulc ulprog -l -cp newprog
```

Löschen der **User Language**-Programme mit Namen **ulprog** und **newprog** sowie der **User Language**-Libraries, deren Namen mit **test** beginnen und auf **lib** enden, aus der Datei **ulcprog.vdb** im BAE-Programmverzeichnis:

```
ulc -dp ulprog newprog -dl test*lib
```

Erzeugen der **User Language**-Library **libsll** aus der Quelltextdatei **libbae.ulh** (der Optimierer ist aktiviert; ein Listing wird auf die Datei **libbae.lst** ausgegeben):

```
ulc libbae.ulh -cl libsll -l2O
```

Kompilieren aller im aktuellen Verzeichnis enthaltenen Quelltextdateien mit der Extension **.ulc** sowie statisches Linken der generierten Maschinenprogramme mit der **User Language**-Library **libsll** (das Makro **USELIB** wird zur Kontrolle der bedingten Übersetzung definiert; der Optimierer ist aktiviert):

```
ulc *.ulc -Define USELIB -lib libsll -O
```

Generieren der **User Language**-Libraries **libstd** und **stdlib** aus der Quelltextdatei **std.ulh** (der Optimierer ist aktiviert, Warnmeldungen mit Severity Level kleiner oder gleich 2 werden ausgegeben):

```
ulc -w2 -O -cl libstd stdlib -Source std.ulh
```

Generieren der **User Language**-Library **liblay** aus der Quelltextdatei **\baeulc\lay.ulh** mit dynamischen Linken der Library **libstd** (der Optimierer ist aktiviert, der Warning Severity Level ist auf den Standardwert 3 gesetzt, die Compiler-Meldungen werden auf die Reportdatei **genlib.rep** anstelle **ulc.log** ausgegeben):

```
ulc /wO -cl liblay -S \baeulc\lay.uh -dll libstd -log genlib.rep
```

Generieren der **User Language**-Programme **LAYPCR** und **TRACEREP** aus den Quelltextdateien **laypcr.old** und **tracerep.ulc** mit dynamischem Linken der Library **liblay** (der Optimierer ist aktiviert):

```
ulc laypcr.old /dll liblay /cp -O /S tracerep
```

3.2.3 Fehlerbehandlung

Mit der wichtigste Bestandteil des Compilers ist die Fehlerbehandlung. Dies ist darin begründet, dass der Compiler am weitaus häufigsten Quellcodedateien zu bearbeiten hat, die fehlerbehaftet sind oder Redundanzen aufweisen (ein absolut fehlerfreies Programm wird i.d.R. nur einmal übersetzt). Die durch den Compiler ausgegebenen Fehlermeldungen sollen dem Programmierer helfen, das zu entwickelnde Programm bzw. die zu erzeugende Library möglichst schnell in einen fehlerfreien Zustand zu bringen.

Der **User Language Compiler** gibt alle Meldungen auf die Standardausgabe und gleichzeitig auf eine Reportdatei aus. Die Ausgabe auf die Reportdatei erfolgt, da die Liste der Meldungen - insbesondere bei der Übersetzung mehrerer Quellcodedateien - sehr umfangreich werden kann. Der Standardname `ulc.log` der Reportdatei kann mit der Compiler-Option `-log` geändert werden.

Im Folgenden ist eine Liste aller im **User Language Compiler** definierten Meldungen, Warnungen und Fehler aufgeführt. Beim Auftreten von Fehlern kann kein ablauffähiger Maschinencode erzeugt werden. Warnungen weisen den Programmierer darauf hin, dass zwar ablauffähiger Maschinen-Code erzeugt werden kann, dieser aber u.U. mit unerwünschten Nebeneffekten behaftet ist. Wo immer möglich, wird der Meldung in Klammern die Zeilennummer vorangestellt, in der der Fehler lokalisiert wurde; diese Zeilennummer bezieht sich entweder auf die Quellcode-Datei (`Ll`) oder auf den Maschinen-Code (`Lp`).

Den unten aufgeführten Warnmeldungen ist jeweils eine Nummer in eckigen Klammern vorangestellt. Diese Angaben sind nicht Bestandteil der tatsächlich ausgegebenen Warnungen, sondern geben vielmehr den Warning Severity Level an, der mit der Option `-w` mindestens eingestellt sein muss, damit die entsprechende Warnung vom **User Language Compiler** ausgegeben wird. Warnungen, die dem Severity Level 0 zugeordnet sind, werden unabhängig vom aktuell eingestellten Warning Severity Level grundsätzlich ausgegeben.

Allgemeine Meldungen

Bei einem syntaktisch falschen Aufruf des **User Language Compilers** korrekten Aufrufsyntax des Compilers angezeigt, und der Compiler-Lauf wird abgebrochen.

Die folgenden allgemeinen Compiler-Meldungen geben Aufschluss über die Aktionen des Compilers bzw. werden als resümierende Meldungen über den Compiler-Lauf ausgegeben:

```
Loeschen Programme aus "n"...
Programm 'n' geloescht.
Loeschen Libraries aus "n"...
Library 'n' geloescht.
Libraries Laden/Linken...
Quellcodedatei "n" wird kompiliert...
Programm 'n' erfolgreich generiert.
Library 'n' erfolgreich generiert.
Quellcodedatei "n" erfolgreich kompiliert.
e Fehler, w Warnungen.
User Language Compiler-Lauf abgebrochen!
User Language Compiler-Lauf erfolgreich beendet.
```

Die folgenden Fehlermeldungen stehen in direktem Zusammenhang und weisen auf fehlende Benutzungsberechtigung zur Ausführung des **User Language Compilers**, ungültige Datei- bzw. Elementnamensangaben, Dateizugriffsprobleme oder Probleme beim Zugriff auf **User Language**-Libraries hin:

```
FEHLER : Die Benutzungsberechtigung fehlt!
FEHLER : Dateiname "n" ist zu lang!
FEHLER : Dateiname "n" enthaelt ungueltige Zeichen!
FEHLER : Elementname 'n' ist zu lang!
FEHLER : Elementname 'n' enthaelt ungueltige Zeichen!
FEHLER : Fehler beim Schreiben auf Listing-Datei "n"!
FEHLER : ULC-Log-Datei "n" kann nicht angelegt werden!
FEHLER : Zu viele Quellcodedateien spezifiziert!
FEHLER : Quellcodedatei "n" nicht gefunden!
FEHLER : Library 'n' nicht gefunden"!
FEHLER : User Language Library 'n' Version inkompatibel!
```

Die folgenden Warnmeldungen stehen in direktem Zusammenhang mit dem Compileraufruf und weisen auf Probleme beim Lesen von Verzeichnissen bzw. beim Zugriff auf Programme oder Libraries, sowie auf Kompatibilitätsprobleme beim Linken von Libraries hin:

```
[0] WARNUNG : Verzeichnis 'n' nicht gefunden/nicht verfuegbar!
[0] WARNUNG : Programm 'n' nicht gefunden!
[0] WARNUNG : Library 'n' nicht gefunden!
[0] WARNUNG : User Language Library 'n' nicht optimiert!
```

Interne Compiler-Fehler

Die folgenden internen Compiler-Fehler können in Zusammenhang mit der Speicherverwaltung stehen oder auf Implementierungs-Lücken im Compiler hinweisen:

```
(Ll) FEHLER : Listen-Ueberlauf!
(Ll) FEHLER : Zu wenig Speicherplatz!
(Ll) FEHLER : INTERNER FEHLER IN function -- BITTE MELDEN!
```

Parser Fehler

Die folgenden Fehler können beim Zugriff auf Quelltextdateien bzw. bei der Syntaxanalyse auftreten:

```
(Ll) FEHLER : Eingabedatei "n" nicht gefunden!
(Ll) FEHLER : Eingabedatei "n" kann nicht gelesen werden!
(Ll) FEHLER : Eingabedatei Dateielement zu lang ('s')!
(Ll) FEHLER : Eingabedatei Ausdruck zu komplex ('s')!
(Ll) FEHLER : Syntaxfehler bei 'string' (unerwartetes Symbol)!
(Ll) FEHLER : Allgemeiner Analyser-Fehler!
```

Semantische Fehler und Warnungen

Die folgenden Fehler können bei der semantischen Prüfung der Quelltextdatei auftreten:

```
(Ll) FEHLER : Identifier 'n' ist zu lang!
(Ll) FEHLER : Character 's' ist zu lang / Kein Character!
(Ll) FEHLER : String 's' ist zu lang!
(Ll) FEHLER : Numerische Angabe 's' ist zu lang!
(Ll) FEHLER : Ungueltige numerische Angabe 's'!
(Ll) FEHLER : Typ 'n' nicht definiert!
(Ll) FEHLER : Typ 'n' mehrfach definiert!
(Ll) FEHLER : Funktion 'n' undefiniert!
(Ll) FEHLER : Funktion 'n' mehrfach definiert!
(Ll) FEHLER : Funktion 'n' ist als System-Funktion definiert!
(Ll) FEHLER : Funktions-Parameter 'n' nicht definiert!
(Ll) FEHLER : Funktions-Parameter 'n' mehrfach definiert!
(Ll) FEHLER : Funktions-Parameter 'n' mehrfach deklariert!
(Ll) FEHLER : Variable 'n' nicht definiert!
(Ll) FEHLER : Variable 'n' mehrfach definiert!
(Ll) FEHLER : Zuweisung an Konstante oder Ergebnis nicht erlaubt!
(Ll) FEHLER : Kein Array; Index-Zugriff nicht moeglich!
(Ll) FEHLER : Ungueltige Array-Index-Angabe!
(Ll) FEHLER : Array-Index nicht im gueltigen Bereich!
(Ll) FEHLER : Zugriff auf Element ('n') aus unbekannter Struktur!
(Ll) FEHLER : Struktur 'n' unbekannt/ungueltig!
(Ll) FEHLER : Struktur 'n' mehrfach definiert!
(Ll) FEHLER : Struktur-Element 'n' unbekannt/ungueltig!
(Ll) FEHLER : Struktur-Element 'n' mehrfach definiert!
(Ll) FEHLER : Index 'n' unbekannt/ungueltig!
(Ll) FEHLER : Index-Variable 'n' unbekannt/ungueltig!
(Ll) FEHLER : 'n' ist keine Variable vom Typ index!
(Ll) FEHLER : 'forall'-Index nicht in 'of'-Index 'n' enthalten!
(Ll) FEHLER : 'continue' nicht innerhalb Schleife!
(Ll) FEHLER : 'break' nicht innerhalb Schleife oder 'switch'!
(Ll) FEHLER : 'void'-Funktion 'n' kann keinen 'return'-Wert liefern!
(Ll) FEHLER : Funktion 'n' muss gueltigen 'return'-Wert liefern!
(Ll) FEHLER : 'return'-Ausdruck nicht typ-kompatibel zur Funktion 'n'!
(Ll) FEHLER : Ausdruck nicht typ-kompatibel zum Parameter 'n'!
(Ll) FEHLER : Ausdruck nicht typ-kompatibel zur Variablen 'n'!
(Ll) FEHLER : Operand nicht typ-kompatibel zum 'n'-Operator!
(Ll) FEHLER : Operanden nicht typ-kompatibel zum 'n'-Operator!
(Ll) FEHLER : Zuweisung an aktiven Schleifen-Index unzulaessig!
(Ll) FEHLER : Ungueltiger 'n'-Ausdruck!
(Ll) FEHLER : Unbekannte/undefinierte Funktion 'n'!
(Ll) FEHLER : Funktion 'n' - zu wenig Parameter spezifiziert!
(Ll) FEHLER : Funktion 'n' - Parameter nicht kompatibel!
(Ll) FEHLER : Funktion 'n' - Parameter nicht im Wertebereich!
(Ll) FEHLER : Ungueltige '#if-#else-#endif'-Konstruktion!
(Ll) FEHLER : Identifier 'n' ist als Makro definiert!
(Ll) FEHLER : Zugriff auf void Makro 'n'!
(Ll) FEHLER : BNF kann nicht in UL-Library gespeichert werden!
(Ll) FEHLER : BNF mehrfach definiert!
(Ll) FEHLER : BNF-Symbol 'n' unbekannt/undefiniert!
(Ll) FEHLER : BNF-Produktion 'n' mehrfach definiert!
(Ll) FEHLER : BNF-Reduce/Reduce-Konflikt bei Produktion 'n'!
(Ll) FEHLER : BNF-Terminalsymbol 'n' ist ungueltig!
(Ll) FEHLER : BNF-Kommentarbegrenzer 's' ist ungueltig!
(Ll) FEHLER : BNF-Funktion 'n' ist nicht vom Type 'int'!
(Ll) FEHLER : Division durch Null!
(Ll) FEHLER : Endlos-Schleife!
(Ll) FEHLER : Funktion 'n' - rekursiver Aufruf!
(Lp) FEHLER : Stack Ueberlauf!
FEHLER : Dateiende erreicht; '}' wurde erwartet!
FEHLER : Inkompatible Index-/Funktions-Referenz(en)!
```

Die folgenden Warnungen können bei der semantischen Prüfung der Quellcodedatei auftreten:

```
[1] (Ll) WARNUNG : BNF enthaelt keine gueltigen Produktionen!
[2] (Ll) WARNUNG : Funktion 'n' - Default-'return'-Wert verwendet!
[1] (Ll) WARNUNG : Funktion 'n' - zu viele Parameter spezifiziert!
[2] (Ll) WARNUNG : Funktion 'n' - Aenderung n. Parameter wird ignoriert!
[2] (Ll) WARNUNG : Funktion 'n' - Aenderung fuer Parameter 'n' wird ignoriert!
[2] (Ll) WARNUNG : Konstanter 'n'-Ausdruck!
[2] (Ll) WARNUNG : Ausdruck hat keine Seiten-Effekte!
[2] (Ll) WARNUNG : Funktion 'n', lokale Variable 'n' verdeckt globale Variable!
[2] (Ll) WARNUNG : Funktion 'n', Parameter 'n' verdeckt globale Variable!
[4] (Ll) WARNUNG : Variable 'n' wurde nicht initialisiert!
[4] (Ll) WARNUNG : Makro 'n' neu definiert!
```

Optimierer-Warnungen

Die folgenden Warnungen werden ggf. vom Optimierer erzeugt und weisen auf Redundanzen im Quellcode hin:

```
[1] (Ll) WARNUNG : BNF ist nicht referenziert!
[2] (Ll) WARNUNG : Globale Variable 'n' ist nicht referenziert!
[2] (Ll) WARNUNG : Funktion 'n' ist nicht referenziert!
[2] (Ll) WARNUNG : Statement wird nicht erreicht!
[3] (Ll) WARNUNG : Funktion 'n', Lokale Variable 'n' ist nicht referenziert!
[3] (Ll) WARNUNG : Funktion 'n', Parameter 'n' ist nicht referenziert!
[4] WARNUNG : Library-Funktion 'n' ist nicht referenziert!
[4] WARNUNG : Library-Variable 'n' ist nicht referenziert!
[4] WARNUNG : Dynamic Link Library 'n' ist nicht referenziert!
```

Datenbank-Zugriffs-Fehler

Die folgenden Fehler können beim Speichern des Maschinen-Codes auftreten:

```
FEHLER : Datenbankdatei "n" kann nicht angelegt werden!
FEHLER : Schreib-/Lesefehler beim Zugriff auf Datei "n"!
FEHLER : Zu viele offene Dateien im System!
FEHLER : Datei "n" ist keine Datenbank/DDB-Datei!
FEHLER : Die Datenbankstruktur in Datei "n" ist beschaedigt!
FEHLER : Der Dateiaufbau ist fehlerhaft in Datei "n"!
FEHLER : Funktion fuer altes Format nicht verfuegbar!
FEHLER : Datenbank Limit ueberschritten!
FEHLER : Datei "n" ist zur Programmversion inkompatibel!
FEHLER : Element 'n' nicht gefunden!
FEHLER : Element 'n' existiert bereits!
FEHLER : Datei "n" nicht gefunden!
FEHLER : Record-Ende erreicht!
FEHLER : Allgemeiner Datenbankfehler!
```

3.3 Interpreter

Der **Bartels User Language Interpreter** ist in verschiedenen Programmumgebungen integriert. Im **Bartels AutoEngineer** sind dies der **Schaltplaneditor**, der **Layouteditor**, der **Autorouter**, der **CAM-Prozessor**, das **CAM-View**-Modul sowie der **Chipeditor**. Aus all diesen Programmteilen können mit Hilfe des **User Language Interpreter User Language**-Programme gestartet werden.

3.3.1 Arbeitsweise

Die Aktivierung des **User Language Interpreter** erfolgt durch den Aufruf eines **User Language**-Programms aus einer der gültigen Interpreterumgebungen. Die Abarbeitung eines **User Language**-Programmaufrufs durch den **User Language Interpreter** erfolgt in den nachfolgend beschriebenen Arbeitsschritten.

Programm laden, Dynamic Link

Zunächst muss der **User Language Interpreter** das **User Language**-Maschinenprogramm über den beim Aufruf spezifizierten Programmnamen aus der Datei `ulcprog.vdb` (im BAE-Programmverzeichnis) laden (Program Load). Beim Laden des Maschinenprogramms führt der Interpreter eine Kompatibilitätsprüfung durch; eine Konsistenzprüfung ist hierbei nicht mehr nötig, da diese zeitaufwändige Arbeit bereits durch den Compiler erledigt wurde. Ein zu ladendes **User Language**-Programm gilt in der aktuellen Interpreterumgebung als kompatibel und damit ablauffähig, wenn die **User Language Compiler**-Version, mit der das Programm erzeugt wurde, identisch mit der des **User Language Interpreter** ist, und wenn das Programm keine Referenzen auf Index-Variablen-Typen oder Systemfunktionen enthält, die in der aktuellen Interpreterumgebung nicht implementiert sind.

User Language-Programme können Anforderungen zur Laufzeiteinbindung von **User Language**-Libraries enthalten (Dynamic Link Requests). Die Einbindung des Maschinencodes der benötigten Dynamic Link Libraries (DLLs) erfolgt automatisch während der Programmladephase. Grundvoraussetzung für die erfolgreiche Durchführung des Dynamic-Link-Prozesses ist die Verfügbarkeit der benötigten Libraries. Außerdem müssen die im Maschinencode der einzubindenden Dynamic Link Libraries enthaltenen Definitionen (Variablen, Funktionen, Funktionsparameter) mit dem bei der Kompilierung durch den **User Language Compiler** überprüften Maschinencode übereinstimmen, damit bei der Ausführung des Programms nicht etwa fehlende oder falsche Libraryobjekte referenziert werden können (was zu einem undefinierten Verhalten des **User Language Interpreter** oder sogar zum Programmabsturz mit möglichem Datenverlust führen könnte). Der im **User Language Interpreter** integrierte Linker führt entsprechende Kompatibilitätsprüfungen durch und verweigert im Falle von Inkonsistenzen die Programmausführung mit der Fehlermeldung `Inkompatible Index-/Funktions-Referenz(en)!`. In solchen Fällen ist das auszuführende **User Language**-Programm neu zu kompilieren.

Programmausführung

Nachdem das **User Language**-Maschinenprogramm geladen (und dynamisch gelinkt) ist, beginnt der Interpreter mit der Ausführung des Programms (Program Execute). Die Programmausführung entspricht dabei der Simulation der **User Language**-Maschinenarchitektur durch Abarbeitung der Instruktionen des Maschinenprogramms. Die Abarbeitung beginnt bei der ersten Instruktion des Maschinenprogramms und ist beendet, wenn der Programmzähler auf eine nicht mehr existente Instruktion des Maschinenprogramms verweist.

Programmterminierung

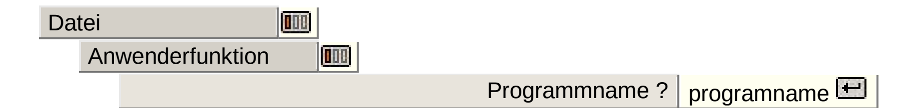
Nach der Ausführung des Maschinenprogramms muss eine Terminierung durchgeführt werden. Dabei wird zunächst der während des Programmlaufs belegte Speicher wieder freigegeben (Program Cleanup), und anschließend wird das Programm selbst entladen (Program Unload).

3.3.2 Programmaufruf

Beim Aufruf eines **User Language**-Programms aus einer der definierten Interpreterumgebungen ist in jedem Fall der Name des auszuführenden Programms zu spezifizieren. Dies kann entsprechend den nachfolgend beschriebenen Methoden entweder explizit oder implizit erfolgen.

Aufruf über Menü

Zum expliziten Aufruf von **User Language**-Programmen ist die Funktion Anwenderfunktion aus dem Menü Datei zu benutzen. Anwenderfunktion aktiviert einen Dialog zur Programmnamensabfrage. Der Programmname kann entweder direkt über Tastatur eingegeben oder nach Anwahl der Schaltfläche Liste aus der Liste der aktuell verfügbaren Programme selektiert werden:



Wird auf die Abfrage nach dem Programmnamen ein Fragezeichen (**?**) eingegeben oder eine Maustaste aktiviert, dann zeigt das System ein Popupmenü mit den Namen aller aktuell in `ulcprog.vdb` verfügbaren **User Language**-Programme an; hieraus kann das zu startende Programm über Mauspick selektiert werden.

Aufruf über Tastatur

Eine Möglichkeit des impliziten Programmaufrufs besteht durch den Programmaufruf über die Tastatur. Hierzu muss sich der Anwender in der Menüleiste der Interpreterumgebung befinden, d.h. diese Art des Programmaufrufs ist immer dann möglich, wenn nicht gerade eine andere interaktive Eingabe über Tastatur erwartet wird. Die Spezifikation des Programmnamens erfolgt dabei implizit durch Drücken einer speziellen Taste. Tabelle 3-2 enthält die Liste der definierten Zuordnung von Tasten zu **User Language**-Programmnamen innerhalb der unterschiedlichen Interpreterumgebungen. Durch Drücken einer der in dieser Tabelle angegebenen Tasten wird - sofern verfügbar - automatisch das entsprechend benannte **User Language**-Programm gestartet. Mit **bae_*** benannte Programme haben dabei Priorität vor den Programmen mit modulspezifischen Namen.

Tabelle 3-2: Tastaturgesteuerter Programmaufruf

	Interpreterumgebung / Programmname						
Tastenbezeichnung	**BAE**	**SCM**	**GED**	**AR**	**CAM**	**CV**	**CED**
Funktionstaste F1	bae_f1	scm_f1	ged_f1	ar_f1	cam_f1	cv_f1	ced_f1
Funktionstaste F2	bae_f2	scm_f2	ged_f2	ar_f2	cam_f2	cv_f2	ced_f2
Funktionstaste F:	bae_f:	scm_f:	ged_f:	ar_f:	cam_f:	cv_f:	ced_f:
Funktionstaste F12	bae_f12	scm_f12	ged_f12	ar_f12	cam_f12	cv_f12	ced_f12
Zifferntaste 0	bae_0	scm_0	ged_0	ar_0	cam_0	cv_0	ced_0
Zifferntaste 1	bae_1	scm_1	ged_1	ar_1	cam_1	cv_1	ced_1
Zifferntaste :	bae_:	scm_:	ged_:	ar_:	cam_:	cv_:	ced_:
Zifferntaste 9	bae_9	scm_9	ged_9	ar_9	cam_9	cv_9	ced_9
Standardtaste a	bae_a	scm_a	ged_a	ar_a	cam_a	cv_a	ced_a
Standardtaste b	bae_b	scm_b	ged_b	ar_b	cam_b	cv_b	ced_b
Standardtaste c	bae_c	scm_c	ged_c	ar_c	cam_c	cv_c	ced_c
Standardtaste :	bae_:	scm_:	ged_:	ar_:	cam_:	cv_:	ced_:

Ereignisgesteuerter Programmaufruf

Eine spezielle Möglichkeit des impliziten Programmaufrufs besteht im ereignisgesteuerten Aufruf von **User Language**-Programmen. Dabei lösen spezielle Ereignisse bzw. Operationen (BAE-Modulstart, Laden bzw. Speichern eines Elements, Ändern des Zoomfaktors oder Selektion eines Toolbarelements) automatisch den Aufruf von **User Language**-Programmen mit definiertem Namen aus. Tabelle 3-3 enthält die Zuordnung der vordefinierten **User Language**-Programmnamen zu den entsprechenden Ereignissen bzw. Operationen der einzelnen Interpreterumgebungen. Mit **bae_*** benannte Programme haben dabei Priorität vor den Programmen mit modulspezifischen Namen. Der Aufruf über die Startupsequenz der Interpreterumgebung eignet sich besonders zur automatischen Voreinstellung von Parametern sowie zur Tastaturprogrammierung und Menübelegung (siehe auch unten). Der implizite Aufruf von **User Language**-Programmen nach dem Laden bzw. vor dem Speichern von Elementen ermöglicht die automatische Aktivierung elementspezifischer Bearbeitungsparameter wie z.B. des zuletzt selektierten Zoombereichs oder spezieller Farbeinstellungen.

Tabelle 3-3: Ereignisgesteuerter Programmaufruf

	Interpreterumgebung / Programmname						
Ereignis	**BAE**	**SCM**	**GED**	**AR**	**CAM**	**CV**	**CED**
Bei BAE-Modulstart	**bae_st.ulc**	**SCM_ST**	**GED_ST**	**AR_ST**	**CAM_ST**	**CV_ST**	**CED_ST**
Vor BAE-Modulende	**bae_exit.ulc**	**SCM_EXIT**	**GED_EXIT**	**AR_EXIT**	**CAM_EXIT**	**CV_EXIT**	**CED_EXIT**
Nach Element laden/schließen	**bae_load.ulc**	**SCM_LOAD**	**GED_LOAD**	**AR_LOAD**	**CAM_LOAD**	**CV_LOAD**	**CED_LOAD**
Nach Element erzeugen	**bae_new.ulc**	**SCM_NEW**	**GED_NEW**	**AR_NEW**	**CAM_NEW**	**CV_NEW**	**CED_NEW**
Bevor Element speichern	**bae_save.ulc**	**SCM_SAVE**	**GED_SAVE**	**AR_SAVE**	**CAM_SAVE**	**CV_SAVE**	**CED_SAVE**
Nach Element speichern	**bae_savd.ulc**	**SCM_SAVD**	**GED_SAVD**	**AR_SAVD**	**CAM_SAVD**	**CV_SAVD**	**CED_SAVD**
Bei Dialogaktivierung	**bae_dial.ulc**	**SCM_DIAL**	**GED_DIAL**	**AR_DIAL**	**CAM_DIAL**	**CV_DIAL**	**CED_DIAL**
Bei Toolbarselektion	**bae_tool.ulc**	**SCM_TOOL**	**GED_TOOL**	**AR_TOOL**	**CAM_TOOL**	**CV_TOOL**	**CED_TOOL**
Bei Zoomfaktoränderung	**bae_zoom.ulc**	**SCM_ZOOM**	**GED_ZOOM**	**AR_ZOOM**	**CAM_ZOOM**	**CV_ZOOM**	**CED_ZOOM**
Bei Mausinteraktion (Drücken linke Maustaste)	**BAE_MS**	**scm_ms.ulc**	**ged_ms.ulc**	**ar_ms.ulc**	**cam_ms.ulc**	**cv_ms.ulc**	**ced_ms.ulc**
Bei Rahmenauswahl mit der Maus	**bae_rect.ulc**	**SCM_RECT**	**GED_RECT**	**AR_RECT**	**CAM_RECT**	**CV_RECT**	**CED_RECT**
Nach Symbol-/Bauteilplatzierung	**BAE_PLC**	**scm_plc.ulc**	**ged_plc.ulc**	**AR_PLC**			**CED_PLC**
Nach Gruppenladeoperationen	**BAE_GRPL**	**scm_grpl.ulc**	**GED_GRPL**				**CED_GRPL**
Bei eingehender Meldung ***(BAE HighEnd)***	**BAE_MSG**	**scm_msg.ulc**	**ged_msg.ulc**	**AR_MSG**	**CAM_MSG**	**CV_MSG**	**CED_MSG**

Tastaturprogrammierung und Menübelegung

Mit der **Bartels User Language** werden Systemfunktionen zur Tastaturprogrammierung und Menübelegung zur Verfügung gestellt. Damit ist es möglich, **User Language**-Programmaufrufe oder BAE-Menüfunktionen auf die Tastatur zu legen (z.B. Taste m zur Aktivierung des **User Language**-Programms **MIRRON** oder Taste U zur Aktivierung der BAE-Menüfunktion Undo). Durch die Zuweisung spezieller **User Language**-Programmaufrufe oder Menüfunktionen auf neue bzw. bestehende Menüs oder Menüeinträge können die Menüoberflächen der **AutoEngineer**-Module mit integriertem **User Language Interpreter** nach Belieben konfiguriert werden. Die Tastaturprogrammierung bzw. die Menübelegung lässt sich vollautomatisch über die jeweiligen **User Language**-Startupprogramme durchführen (siehe hierzu das mit der BAE-Software ausgelieferte **User Language**-Programm **UIFSETUP** welches indirekt über die in Tabelle 3-3 aufgeführten Startupprogramme aufgerufen wird). Mit einem geeigneten **User Language**-Programm ist sogar die dynamische Änderung der Tastatur- und Menübelegung (d.h. *online* während der Arbeit im **AutoEngineer**) möglich. Derartige Features sind z.B. in dem mit der BAE-Software ausgelieferten **User Language**-Programm **KEYPROG** implementiert. Damit besteht völlige Freiheit bei der Konfiguration der Benutzeroberflächen der **AutoEngineer**-Module mit integriertem **User Language Interpreter**.

Menüfunktionstastenbelegung

In BAE-Pulldownmenüoberflächen unterstützt das über Taste **5** aufrufbare **User Language**-Programm **KEYPROG** die direkte Zuweisung von Menüfunktionen zu Tasten. Dazu ist nach Selektion der zu programmierenden Taste über Tastaturprogrammierung und HotKeys belegen und Anklicken der Schaltfläche Menuauswahl im Programmauswahlmenü einfach die gewünschte Menüfunktion zu selektieren.

Definition von Bearbeitungssequenzen (Makros)

Über Anwenderfunktion, in **User Language**-Programmaufrufen über die Funktion **ulsystem**, bei der Definition von Tasten und Menüfunktionen in der Datei `bae.ini` sowie bei der Online-Tastenprogrammierung mit dem **User Language**-Programm **KEYPROG** kann anstatt eines einfachen **User Language**-Programmnamens eine komplette Bearbeitungssequenz angegeben werden. Die einzelnen Bearbeitungsschritte sind durch das Zeichen `:` zu trennen. Ein **User Language**-Programmname, der nicht am Anfang der Bearbeitungssequenz steht, ist durch ein vorangestelltes `p` zu kennzeichnen. Ein vorangestelltes `#` referenziert direkt eine interne BAE-Funktion entsprechend dem Aufruf von **bae_callmenu**. In einfachen Anführungszeichen gesetzte Texte werden als Texteingaben übernommen. Mit `t` wird auf die Eingabe eines Textes durch den Benutzer gewartet. `s` wartet auf eine Menuselektion, gefolgt von `l`, `m` oder `r` und einem Menüindex (Start der Zählung bei 0) wird eine Menuselektion entsprechend dem Aufruf von **bae_storemenuiact** durchgeführt. Ein alleinstehendes `m` wartet auf einen Mausklick. Folgt darauf ein `l`, `m` oder `r`, dann wird ein Mausklick mit der entsprechenden Maustaste an der zu Beginn der Bearbeitungssequenz ermittelteten Mausposition durchgeführt. Folgen der Maustaste noch durch Komma getrennte Koordinatenangaben, so wird die Maustaste an der angegebenen Position aktiviert.

Damit ist es möglich, Untermenüpunkte, wie z.B. Symbol/Label Query (Bearbeitungssequenz `scmpart:s5:m:t:mr`) direkt auf Tasten zu legen. Auch ganze Abfolgen von immer wiederkehrenden Arbeitsschritten sind möglich. So wird z.B. im **Schaltplaneditor** mit der Sequenz `#500:ml:mr:sl3:'4':'0':mr:sl0` von der aktuellen Mausposition aus ein waagerechtes 4mm langes Stück Grafiklinie nach rechts gezeichnet, wie man es immer wieder bei der Bearbeitung von Symbolen zur grafischen Anbindung von Pins an eine Symbolbox benötigt.

Benutzerspezifische Programmeinstellungen und Menükonfiguration

Zur vereinfachten Handhabung von benutzerspezifischen Einstellungen können Parameterdefinitionen, Tastaturbelegungen und Menüerweiterungen in die Datei **`bae.ini`** im BAE-Programmverzeichnis eingetragen werden. Die Definitionen aus dieser Datei werden einmalig beim BAE-Programmstart in den Speicher geladen und können anschließend mit der **User Language**-Systemfunktion **varget** abgefragt werden.

Die mit der BAE-Software ausgelieferten **User Language**-Programme berücksichtigen relevante Definitionen und Parametereinstellungen aus **`bae.ini`**. Zur Aktivierung von Änderungen in **`bae.ini`** ist somit lediglich ein Neustart des von den Änderungen betroffenen BAE-Programmmoduls erforderlich. Benutzerspezifische Parametereinstellungen können damit komfortabel zwischen verschiedenen BAE-Versionen bzw. BAE-Installationen ausgetauscht werden.

Die Datei **`bae.ini`** ist in Sektionen für die einzelnen BAE-Module unterteilt. Beim Modulstart werden nur die Einstellungen des aktiven Moduls berücksichtigt. Die Definitionen aus der Sektion **`std`** gelten für alle Module.

Neben einfachen Parameterzuweisungen können mit den Schlüsselwörtern **`key`** und **`fkey`** Standard- und Funktionstasten mit Bearbeitungssequenzen belegt werden. Die Schlüsselwörter **`addmenu`**, **`addmenuitem`**, **`addsmenuitem`** und **`addioitem`** erlauben die Erweiterung der Menüstruktur. Es können jeweils nur Menüpunkte an das Ende eines Hauptmenüs oder des Import/Export-Menüs angehängt werden. Ein Einfügen von Menüpunkten in Menüs ist nicht möglich, da dadurch die Topic-Zuweisung für die Online-Hilfe durcheinander geraten würde.

Die Syntax der einzelnen Kommandos ist aus der Inlinedokumentation der mitgelieferten Datei **`bae.ini`** zu ersehen. In ihrer Originalversion definiert diese Datei die Parametereinstellungen aus den mit der BAE-Software ausgelieferten **User Language**-Programmen, also die Standardeinstellungen, die auch dann verwendet werden, wenn die Datei **`bae.ini`** beim BAE-Programmaufruf nicht im BAE-Programmverzeichnis verfügbar ist.

3.3.3 Fehlerbehandlung

Während der Bearbeitung eines **User Language**-Programmaufrufs durch den **User Language Interpreter** können Fehler auftreten. Diese Fehler werden der Interpreterumgebung zurückgemeldet, und es wird ggf. eine entsprechende Fehlermeldung in der Mitteilungszeile der Interpreterumgebung angezeigt. Die möglichen Fehlermeldungen sind nachfolgend aufgelistet.

Interne Interpreter-Fehler

Die folgenden internen Interpreter-Fehler beziehen sich entweder auf die Speicherverwaltung oder weisen auf Implementierungslücken im Interpreter selbst hin; diese Fehler sind so schwerwiegend (Fatal Errors), dass auch die Interpreterumgebung abgebrochen werden muss:

```
FEHLER : Zu wenig Speicherplatz!
FEHLER : Allgemeiner User Language-Interpreter-Fehler!
FEHLER : INTERNER FEHLER -- BITTE MELDEN!
```

Programm-Lade-Fehler

Die folgenden Fehler können beim Laden des **User Language**-Programms auftreten:

```
FEHLER : Programm 'n' bereits geladen (rekursiver Aufruf)!
FEHLER : Programm 'n' nicht gefunden!
FEHLER : User Language-Programm-Version inkompatibel!
FEHLER : Inkompatible Index-/Funktions-Referenz(en)!
```

Programm-Laufzeit-Fehler

Die folgenden Fehler können während der Ausführung des **User Language**-Programms, d.h. zur Laufzeit des Programms auftreten; der in der Fehlermeldung angezeigte Programmzähler (`PCl`) gibt dabei die Zeile im **User Language**-Maschinenprogramm an, in der der Fehler aufgetreten ist (sofern durch den **User Language Compiler** eine entsprechende Listing-Datei für das betreffende Programm erzeugt wurde, lässt sich mit deren Hilfe die entsprechende Zeile im Quellcode ermitteln):

```
(PCl) FEHLER : Stack Unterlauf (Programm-Struktur beschaedigt)!
(PCl) FEHLER : Stack Ueberlauf!
(PCl) FEHLER : Division durch Null!
(PCl) FEHLER : Funktions-Aufruf fehlgeschlagen!
(PCl) FEHLER : System-Funktion in dieser Umgebung nicht verfuegbar!
(PCl) FEHLER : System-Funktion nicht implementiert!
(PCl) FEHLER : User-Funktion nicht gefunden!
(PCl) FEHLER : Referenzierte Funktion ist vom falschen Typ!
(PCl) FEHLER : Ungueltige Parameter fuer referenzierte Funktion!
(PCl) FEHLER : Fehler beim Zugriff auf Array!
(PCl) FEHLER : Ungueltiger Array-Index!
(PCl) FEHLER : Datei-Zugriffs-Fehler!
(PCl) FEHLER : Datei-Lese-Fehler!
(PCl) FEHLER : Datei-Schreib-Fehler!
```

Datenbank-Zugriffs-Fehler

Die folgenden Fehler können beim Zugriff auf die Design Datenbank (DDB) des **Bartels AutoEngineer** auftreten; treten derartige Fehler beim Laden von **User Language**-Programmen auf, dann liegt unter Umständen eine fehlerhafte Installation der **Bartels AutoEngineer** Software vor; treten Datenbank-Zugriffsverletzungen hingegen während der Programmlaufzeit auf, dann bezieht sich die entsprechende Fehlermeldung in aller Regel auf Implementierungs-Fehler im abzuarbeitenden **User Language**-Programm:

```
FEHLER : Datenbankdatei 'n' kann nicht angelegt werden!
FEHLER : Schreib-/Lesefehler beim Zugriff auf Datei 'n'!
FEHLER : Zu viele offene Dateien im System!
FEHLER : Datei 'n' ist keine Datenbank/DDB-Datei!
FEHLER : Die Datenbankstruktur in Datei 'n' ist beschaedigt!
FEHLER : Der Dateiaufbau ist fehlerhaft in Datei 'n'!
FEHLER : Funktion fuer altes Format nicht verfuegbar!
FEHLER : Datenbank Limit ueberschritten!
FEHLER : Datei 'n' ist zur Programmversion inkompatibel!
FEHLER : Element 'n' nicht gefunden!
FEHLER : Element 'n' existiert bereits!
FEHLER : Datei 'n' nicht gefunden!
FEHLER : Record-Ende erreicht!
FEHLER : Allgemeiner Datenbankfehler!
```

Kapitel 4
BAE User Language-Programme

Dieses Kapitel enthält eine Übersicht über die mit dem **Bartels AutoEngineer** ausgelieferten **User Language**-Includedateien und **User Language**-Programme sowie Hinweise zur Bereitstellung der Programme im **Bartels AutoEngineer**. Bei der Auflistung der **User Language**-Programme wurde eine Gliederung nach möglichen Anwendungsgebieten bzw. Interpreterumgebungen vorgenommen.

Inhalt

4.1 User Language-Includedateien

Die in diesem Abschnitt aufgeführten **User Language**-Programmdateien sind Includedateien, die häufig benötigte Definitionen und Funktionen enthalten. Diese Include-Files werden von den mit der BAE-Software ausgelieferten **User Language**-Programmen extensiv referenziert.

4.1.1 Standard-Includedateien

std.ulh (STD) -- Standard-Include

Die Definitionen und Deklarationen aus der Includedatei **std.ulh** sind kompatibel zu allen Interpreterumgebungen des **Bartels AutoEngineer** (also sowohl zum **Schaltplaneditor**, als auch zum **Layouteditor**, zum **Autorouter**, zum **CAM-Prozessor**, zum **CAM-View**-Modul und zum **Chipeditor**). **std.ulh** enthält neben häufig benötigten Konstanten unter anderem auch Funktionen zur Bilddarstellung und Datenkonvertierung, Routinen für das Fehlermeldewesen und zur Internationalisierung sowie BAE-Menüfunktionen und Utilities zur Abfrage der BAE-Softwarekonfiguration und zur Manipulation des Arbeitsbereichs.

baeparam.ulh (STD) -- BAE-Parameterzugriff

Die Definitionen und Deklarationen aus der Includedatei **baeparam.ulh** sind kompatibel zu allen Interpreterumgebungen des **Bartels AutoEngineer**. **baeparam.ulh** enthält Funktionen für den Zugriff auf System- und Bearbeitungsparameter. In die Datei **baeparam.ulh** ist über Include-Anweisung die Quellcodedatei **std.ulh** (siehe oben) eingebunden.

pop.ulh (STD) -- Popup-Utilities

Die Definitionen und Deklarationen aus der Includedatei **pop.ulh** sind kompatibel zu allen Interpreterumgebungen des **Bartels AutoEngineer**. In **pop.ulh** sind Routinen zur menügesteuerten Datei- und Elementnamensabfrage, allgemeine Popupfunktionen, Funktionen zum Auflisten von Verzeichnisinhalten sowie BAE-Menüfunktionen und Utilities zur Abfrage der BAE-Softwarekonfiguration definiert. In die Datei **pop.ulh** ist über Include-Anweisung die Quellcodedatei **std.ulh** (siehe oben) eingebunden.

popdraw.ulh (STD) -- Popup-Zeichenfunktionen

Die Definitionen und Deklarationen aus der Includedatei **popdraw.ulh** sind kompatibel zu allen Interpreterumgebungen des **Bartels AutoEngineer**. **popdraw.ulh** enthält allgemeine Utilities und Definitionen zur Anzeige von Icons und Buttons sowie zur Grafikausgabe in Popupmenüs. In die Datei **popdraw.ulh** ist über Include-Anweisung die Quellcodedatei **std.ulh** (siehe oben) eingebunden.

mnu.ulh (STD) -- Menüfunktionen

Die Definitionen und Deklarationen aus der Includedatei **mnu.ulh** sind kompatibel zu allen Interpreterumgebungen des **Bartels AutoEngineer**. **mnu.ulh** enthält erweiterte Popup- und Menü-Utilities zur Textanzeige, zur Textabfrage, zur Farbauswahl, zur Anzeige von BAE-Produktinformationen, usw. In die Datei **mnu.ulh** ist über Include-Anweisung die Quellcodedatei **pop.ulh** (siehe oben) eingebunden.

sql.ulh (STD) -- SQL-Utilities

Die Definitionen und Deklarationen aus der Includedatei **sql.ulh** sind kompatibel zu allen Interpreterumgebungen des **Bartels AutoEngineer**. **sql.ulh** enthält eine Reihe nützlicher Routinen für die Verwaltung von relationalen SQL-Datenbanken sowie den Zugriff auf deren Inhalt. In die Datei **sql.ulh** ist über Include-Anweisung die Quellcodedatei **pop.ulh** (siehe oben) eingebunden.

xml.ulh (STD) -- XML-Utilities

Die Definitionen und Deklarationen aus der Includedatei **xml.ulh** sind kompatibel zu allen Interpreterumgebungen des **Bartels AutoEngineer**. **xml.ulh** enthält eine Reihe nützlicher Import- und Export-Funktionen fuer XML-Dateien. In die Datei **xml.ulh** ist über Include-Anweisung die Quellcodedatei **std.ulh** (siehe oben) eingebunden.

4.1.2 SCM-Includedateien

scm.ulh (SCM) -- SCM-/Schaltplaneditor-Utilities

Die Definitionen und Deklarationen aus der Includedatei **scm.ulh** sind kompatibel zur Interpreterumgebung des **Schematic Editors**. In **scm.ulh** sind unter anderem Funktionen zum Kopieren von SCM-Elementen und zur Zuweisung von Regeln an SCM-Elemente definiert. In die Datei **scm.ulh** ist über Include-Anweisung die Quellcodedatei **std.ulh** (siehe oben) eingebunden.

4.1.3 Layout-Includedateien

lay.ulh (LAY) -- Layout-Utilities

Die Definitionen und Deklarationen aus der Includedatei **lay.ulh** sind kompatibel zu den Interpreterumgebungen des **Layouteditors**, des **Autorouters** und des **CAM-Prozessors**. In **lay.ulh** sind unter anderem Funktionen zur analytischen Geometrie, zur Auswertung von Netzlistendaten, zum Kopieren von Layoutelementen sowie zur Abfrage von Lagenbezeichnungen definiert. **lay.ulh** enthält darüber hinaus auch Funktionen für den Zugriff auf das **Neuronale Regelsystem** sowie die Anwendung desselben in den Layoutmodulen des **Bartels AutoEngineer**. In die Datei **lay.ulh** ist über Include-Anweisung die Quellcodedatei **std.ulh** (siehe oben) eingebunden.

4.1.4 IC-Design-Includedateien

icd.ulh (ICD) -- IC-Design-Utilities

Die Definitionen und Deklarationen aus der Includedatei **icd.ulh** sind kompatibel zur Interpreterumgebung des **IC-Design-Chipeditors**. In **icd.ulh** sind unter anderem Funktionen zur analytischen Geometrie, zum Kopieren von **IC-Design**-Elementen sowie zur Abfrage von Lagenbezeichnungen definiert. **icd.ulh** enthält darüber hinaus auch Funktionen für den Zugriff auf das **Neuronale Regelsystem** sowie die Anwendung desselben im **IC-Design**-System des **Bartels AutoEngineer**. In die Datei **icd.ulh** ist über Include-Anweisung die Quellcodedatei **std.ulh** (siehe oben) eingebunden.

4.2 User Language-Programme

Dieser Abschnitt enthält gegliedert nach Interpreterumgebung bzw. Anwendungsgebieten und alphabetisch sortiert eine Auflistung der mit dem **Bartels AutoEngineer** ausgelieferten **User Language**-Programme. Die hier aufgeführten **User Language**-Programme werden bei der Installation des **Bartels AutoEngineer** in einem speziell hierfür vorgesehenen Verzeichnis abgelegt.

4.2.1 Standard-Programme

Die nachfolgend aufgeführten **User Language**-Programme sind in *allen* Interpreterumgebungen des **Bartels AutoEngineer** (also im **Schaltplaneditor**, im **Layouteditor**, im **Autorouter**, im **CAM-Prozessor**, im **CAM-View**-Modul und im **Chipeditor**) ablauffähig.

ARC (STD) -- Kreisbogen/Kreis zeichnen

Das Programm **arc.ulc** ermittelt die aktuell aktive BAE-Menüfunktion und führt (sofern zulässig bzw. möglich) eine Untermenüfunktion zum schnellen Zeichnen eines Kreises oder Kreisbogens aus. Bei Aufruf ohne aktuell aktive BAE-Menüfunktion wird ein Dialog zur Auswahl des Bearbeitungsmodus aktiviert. Die Funktionsfähigkeit dieses Programms ist nur gewährleistet, wenn es implizit über Tastendruck (z.B. c oder C) aufrufbar ist.

BAE_DIAL (STD) -- BAE Dialogbox-Aktion

Das Programm **bae_dial.ulc** wird automatisch aktiviert wenn ein Dialogboxelement mit registriertem Aktionscode selectiert wird. Die Aktivierungsdaten der Dialogbox werden von der Interaktionsqueue auf globale Variablen übertragen.

BAE_EXIT (STD) -- BAE Programmende-Aktion

Das Programm **bae_exit.ulc** wird automatisch vor Beednigung des aktuell aktiven BAE-Programm-Moduls aktiviert. **bae_exit.ulc** hebt alle aktuell aktiven Sperren für das aktuell geladene Element auf.

BAE_LOAD (STD) -- BAE Lade-/Schließen-Aktion

Das Programm **bae_load.ulc** wird automatisch nach dem Laden, Erzeugen oder Schließen eines Elements aktiviert. **BAE_LOAD** zeigt den Datei- und Elementnamen des geladenen Elements sowie den mit **MSMODE** aktuell selektierten Mausoperationsmodus in der Statuszeile an, sofern das System mit Pulldownmenüs betrieben wird. Beim Schließen des aktuell geladenen Elements wird die Datei- und Elementnamensanzeige gelöscht. Außerdem reaktiviert **BAE_LOAD** den ggf. zuvor mit **BAE_SAVE** gespeicherten, elementspezifischen Modus zur Anzeige der Werkzeugleiste.

BAE_NEW (STD) -- BAE Neues Element-Aktion

Das Programm **bae_new.ulc** wird automatisch nach dem Erzeugen eines Erzeugen eines Elements aktiviert. **bae_new.ulc** nimmt Standardparametereinstellungen für das neue Element vor.

BAE_RECT (STD) -- BAE Mausrahmenaktion

Das Programm **bae_rect.ulc** wird automatisch bei Betätigung der linken Maustaste aktiviert, wenn gerade keine andere Funktion aktiv ist, und im Grafikarbeitsbereich mit der Maus ein Rahmen mit einer Seitenlänge von wenigstens 10 Pixeln aufgezogen wurde. Entsprechend dem mit **MSMODE** aktuell eingestellten Mausoperationsmodus wird eine Gruppenfunktion auf die im aufgezogenen Rechteck platzierten Elemente angewendet.

BAE_SAVD (STD) -- BAE Speichern beendet Aktion

Das Programm **bae_savd.ulc** wird automatisch nach dem Speichern eines Elements aktiviert. **bae_savd.ulc** unterstützt die Ausführung eines benutzerspezifischen **User Language**-Programms.

BAE_SAVE (STD) -- BAE Speicheraktion

Das Programm **bae_save.ulc** wird automatisch vor dem Speichern eines Elements aktiviert. **BAE_SAVE** sichert die aktuellen Entwurfsansichten aus der Toolbar mit dem zu speichernden Element sowie den Modus zur Anzeige der Werkzeugleiste.

BAE_ST (STD) -- BAE Startup

Das Programm **bae_st.ulc** wird automatisch beim Aufruf von BAE-Programm-Modulen aktiviert. **BAE_ST** startet seinerseits das Programm **UIFSETUP** zur Aktivierung einer vordefinierten Menü- und Tastaturbelegung in der aktuellen Interpreterumgebung. Darüber hinaus aktiviert **BAE_ST** je nach aktuell aktiver Interpreterumgebung eines der **User Language**-Programme **SCMSETUP**, **GEDSETUP**, **ARSETUP**, **CAMSETUP**, **CVSETUP** und **CEDSETUP** wodurch modulspezifische Standardparameter wie Eingabe- und Hintergrundraster, Winkel- und Rasterfreigabe, Koordinatenanzeigemodus, Vorzugslage, Mincon-Funktion, usw. gesetzt werden. Alle diese Programme können selbstverständlich leicht an anwender- bzw. projektspezifische Bedürfnisse angepasst werden. **BAE_ST** startet auch ein projektspezifisches **User Language**-Programm wenn eine Projektdatei aktiviert ist, und wenn hierzu eine Datei mit der Endung `.aut` verfügbar ist. Der Name des projektspezifischen **User Language**-Programms wird aus der ersten Zeile der `.aut`-Datei entnommen.

BAE_TOOL (STD) -- BAE Toolbaraktion

Das Programm **bae_tool.ulc** wird automatisch bei Selektion eines Toolbarobjektes aktiviert. **BAE_TOOL** löst in Abhängigkeit vom Eingabestring eine weitere Aktion aus. Hierbei werden Integerangaben als Anforderung zum Aufruf der dem Zahlenwert zugeordneten BAE-Menüfunktion interpretiert, während andere Eingaben den Aufruf eines entsprechend benannten **User Language**-Programms auslösen. Ein durch ein Leerzeichen getrennter String wird automatisch als Parameter an die nachfolgend aufzurufende Funktion übergeben.

BAE_ZOOM (STD) -- BAE Zoomaktion

Das Programm **bae_zoom.ulc** wird automatisch bei einer Änderung des Zoombereichs aktiviert. **BAE_ZOOM** aktualisiert ggf. die Entwurfsansicht in der Toolbar.

BAEMAN (STD) -- BAE Windows Online-Handbuch (Windows)

Das Programm **baeman.ulc** aktiviert den **Windows**-Webbrowser zur Anzeige des **Bartels AutoEngineer**-Benutzerhandbuchs.

BITMAPIN (STD) -- Import Bitmap Data

Das Programm **bitmapin.ulc** importiert Bitmapdaten verschiedener Formate. Bei Übernahme in den Schaltplan werden die Bitmapdaten in Grafikfächen umgewandelt. Bei Übernahme in das Layoutsystem werden die Bitmapdaten in Dokumentarflächen einer angebbaren Dokumentarlage umgewandelt. Die importierten Bitmapdaten werden automatisch zur Gruppe selektiert um eine einfache Weiterbearbeitung (verschieben, skalieren, usw.) zu ermöglichen.

CLOGDEFS (STD) -- Logische Bibliotheksdefinitionen pruefen

Das Programm **clogdefs.ulc** ist ein Utilityprogramm für das Bibliotheksmanagement. Mit **clogdefs.ulc** lassen sich die SCM-Symbolnamen selektierbarer Bibliotheksdateien des aktuellen Verzeichnisses auf das Vorhandensein zugehöriger Logischer Bibliothekseinträge in einer frei wählbaren Layoutbibliothek überprüfen. Das Resultat der Bibliotheksprüfung wird in einem Popupmenü mit Ausgabeoption angezeigt.

CMDCALL (STD) -- Kommandosequenz ausfuehren

Das Programm **cmdcall.ulc** ermittelt die Eingabedaten einer Kommandosequenz und führt diese Kommandosequenz aus. BAE-Menükommandosequenzen sind im Funktionshandbuch beschrieben (siehe brgar.htm, brgcam.htm, brgcv.htm, brgged.htm und brgscm.htm). Dies entspricht grundsätzlich dem Verhalten der Funktion Anwenderfunktion mit dem Unterschied, dass nicht ein separates Dialogfenster sondern das BAE-Fenster selbst den Eingabefokus erhält. Damit können BAE-Operationen ferngesteuert mit Tools wie z.B. **StrokeIT** (Programmsteuerung über Mausgestik) ausgelöst werden. Das Programm **cmdcall.ulc** ist standardmässig der Tastenkombination `Umschalt-Strg-R` zugewiesen. D.h., ein ferngesteuerter Funktionsaufruf kann mit `Umschalt-Strg-R` gefolgt von der Kommandosequenz und Drücken Eingabetaste ausgelöst werden.

COPYELEM (STD) -- DDB-Dateielemente kopieren

Das Programm **copyelem.ulc** erlaubt analog zum Utilityprogramm **COPYDDB** das Kopieren von Elementen von einer DDB-Datei in eine andere DDB-Datei. Dabei können in einer Selektionsbox aus der Quelldatei beliebige viele Elemente der zuvor selektierten Elementklasse zum Kopieren markiert werden. Auch die Angabe von Elementnamensmustern wie z.B. `74ls*` für SCM-Symbole oder `dil*` für Layoutgehäusebauformen ist möglich. Nach Auswahl der zu kopierenden Elemente ist die Zieldatei anzugeben sowie ggf. der Kopiermodus (Alles kopieren zum Überschreiben von existierenden Elementen in der Zieldatei, oder Keine Ersetzungen zur Beibehaltung existierender Elemente in der Zieldatei) zu selektieren. Sofern SCM-Symbole zum Kopieren ausgewählt wurden, können wahlweise auch die zugehörigen logischen Bauteildefinitionen mitkopiert werden.

DBCOPY (STD) -- SQL-Datenbanken kopieren

Mit dem Programm **dbcopy.ulc** können SQL-Tabellenstrukturen und Datenbankeinträge selektierbarer SQL-Datenbankdateien kopiert werden.

DBREPORT (STD) -- SQL-Datenbankreport

Das Programm **dbreport.ulc** dient der Anzeige der SQL-Tabellenstrukturen und Datenbankeinträge selektierbarer SQL-Datenbankdateien.

DELCOLOR (STD) -- Selektierbare Farbpalette loeschen

Mit dem Programm **delcolor.ulc** können selektiv Farbtabellen gelöscht werden.

DELDVINF (STD) -- Entwurfsansicht-Information löschen

Das Programm **deldvinf.ulc** löscht alle Informationen über Entwurfsansichten aus der DDB-Datei des aktuell geladenen Elements.

DESKCALC (STD) -- Taschenrechner

Das Programm **deskcalc.ulc** aktiviert in einem Popupmenü einen Taschenrechner mit Grundrechenarten und trigonometrischen Funktionen.

DIR (STD) -- Verzeichnisinhalt auflisten

Das Programm **dir.ulc** listet den Inhalt des aktuellen Verzeichnisses auf dem Bildschirm auf.

DISPUTIL (STD) -- Display Utilities

Das Programm **disputil.ulc** bietet eine einheitliche Aufrufschnittstelle für häufig benötigte, modulspezifische Bilddarstellungsfunktionen.

DISTANCE (STD) -- Distanzabfrage

Mit dem Programm **distance.ulc** können Distanzabfragen durchgeführt werden. Es werden dabei die absoluten, horizontalen und vertikalen Abstände sowie der Winkel zwischen zwei mausselektierbaren Koordinaten angezeigt. Die Längeneinheiten für die Abstandsanzeige werden zunächst aus dem aktuell eingestellten Koordinatenanzeigemodus ermittelt und können anschließend im Anzeigemenü geändert werden.

DONE (STD) -- Eingabeinteraktion beenden

Das Programm **done.ulc** dient dazu, die Eingabesequenz von Funktionen zur Erstellung bzw. Manipulation von Polygonen, Verbindungen oder Leiterbahnen zu beenden. Die Funktionsfähigkeit dieses Programms ist nur gewährleistet, wenn es implizit über Tastendruck (z.B. `Enter`) aufrufbar ist.

FAVORITE (STD) -- Favoritenmenüverwaltung

Das Programm **favorite.ulc** dient der Konfiguration und Aktivierung eines benutzerdefinierten Favoritenmenüs. Dieses Programm ist zum Aufruf über eine Toolbarschaltfläche vorgesehen.

FILEUTIL (STD) -- Dateibearbeitungsfunktionen

Das Programm **fileutil.ulc** ermöglicht die Aktivierung einer Reihe von Dateiverwaltungsfunktionen wie z.B. das Auflisten von Verzeichnisinhalten, das Umkopieren, Löschen und Auflisten von Dateien, Suchen und Anzeigen von DDB-Elementen, die Anzeige von SQL-Datenbankinhalten, usw.

FILEVIEW (STD) -- Dateiinhalt auflisten

Das Programm **fileview.ulc** listet den Inhalt einer frei wählbaren ASCII-Datei auf dem Bildschirm auf.

FINDELEM (STD) -- DDB-Elemente suchen und anzeigen

Mit dem Programm **findelem.ulc** kann die Festplatte (d.h. angebbare Verzeichnisse mit Unterverzeichnissen) nach DDB-Elementen selektierbarer Datenbankklassen durchsucht werden. Die Namen der zu suchenden Elemente können mit Wildcards angegeben werden. Die gefundenen Elemente werden wahlweise nach Elementnamen oder nach Dateinamen sortiert. Sofern die gewählte Datenbankklasse mit der aktuellen Interpreterumgebung kompatibel ist, wird ein Browser zum Anzeigen bzw. Laden der gefundenen Elemente aktiviert.

GRTOGGLE (STD) -- Eingaberaster freigeben/einhalten

Das Programm **grtoggle.ulc** schaltet den Modus zur Einhaltung bzw. Freigabe des Eingaberasters um. Beim Umschalten auf den Modus zum Einhalten des Rasters wird die Einstellung für die Winkelfreigabe automatisch wiederhergestellt. Das Programm ist für den impliziten Aufruf über Tastatur (z.B. durch Zuweisung auf Hotkey g oder G) gedacht.

HELP (STD) -- Online-Hilfe

Das Programm **help.ulc** ermöglicht die Anzeige von Onlinedokumentation.

HISTORY (STD) -- Aufruf Elementbearbeitungshistorie

Das Programm **history.ulc** ermöglicht den Zugriff auf eine Dateielementhistorie mit einem Auswahlmenü für Ladeoperationen. Dieses Programm ist zum Aufruf über eine Toolbarschaltfläche vorgesehen.

HLPKEYS (STD) -- Online-Hilfe - Tastaturbelegung anzeigen

Das Programm **hlpkeys.ulc** zeigt die aktuell definierte Tastaturbelegung an. Die Anzeige erfolgt in einem Popupmenü mit Ausgabeoption.

HLPPROD (STD) -- Online-Hilfe - BAE-Produktinformation

Das Programm **hlpprod.ulc** zeigt in einem Popupmenü BAE-Produktinformationen wie z.B. die Softwarekonfiguration, die aktivierte Benutzeroberfläche, das aktive Programm-Modul, die Versionsnummer, usw. an.

INFO (STD) -- Info

Das Programm **info.ulc** bietet eine einheitliche Aufrufschnittstelle für häufig benötigte, interpreterspezifische Reportfunktionen. **INFO** ermittelt die aktuelle Interpreterumgebung und aktiviert in Abhängigkeit hiervon ein interpreterspezifisches Reportprogramm (**SCMPCR** im **Schaltplaneditor**, **LAYPCR** im Layoutsystem, **HLPPROD**, wenn kein Element geladen ist, usw.).

INIEDIT (STD) -- bae.ini-Editor

Mit dem Programm **iniedit.ulc** können allgemeine BAE-Systemparameter in der Datei `bae.ini` aus dem BAE-Programmverzeichnis interaktiv editiert werden. Eine Sicherungskopie des alten `bae.ini`-Inhalts wird unter `bae.bak` abgelegt.

KEYPROG (STD) -- User Language-Programmaufruf und Tastaturprogrammierung

Das Programm **keyprog.ulc** stellt Funktionen zum menügesteuerten **User Language**-Programmaufruf, zur Menübelegung und Tastaturprogrammierung sowie zur Erfassung und Pflege einer relationalen Datenbank mit Hilfstexten über **User Language**-Programme bereit.

LARGER (STD) -- Pickelement vergrößern/verbreitern

Das Programm **larger.ulc** ermittelt die aktuell aktive BAE-Menüfunktion und führt (sofern zulässig bzw. möglich) eine Untermenüfunktion zur Vergrößerung bzw. Verbreiterung des aktuell bearbeiteten Objekts durch; die Funktionsfähigkeit dieses Programms ist nur gewährleistet, wenn es implizit über Tastendruck (z.B. b) aufrufbar ist.

LFTOCRLF (STD) -- Konvertierung LF nach CRLF

Das Programm **lftocrlf.ulc** wandelt alle Zeilenvorschub-Zeichen einer frei wählbaren ASCII-Datei in die Steuerzeichensequenz Wagenrücklauf und Zeilenvorschub um.

LIBCONTS (STD) -- Bibliotheksinhalt auflisten

Das Programm **libconts.ulc** ist ein Utilityprogramm für das Bibliotheksmanagement. Mit **libconts.ulc** kann eine Inhaltsübersicht für alle im aktuellen Verzeichnis enthaltenen DDB- und DEF-Dateien erstellt werden. Die so erzeugten Inhaltsverzeichnisse werden in Popupmenüs mit Ausgabeoptionen angezeigt.

LIBCRREF (STD) -- Bibliothekselement-Querverweisliste

Das Programm **libcrref.ulc** ist ein Utilityprogramm für das Bibliotheksmanagement. Mit **libcrref.ulc** kann eine Cross-Referenz für die Bibliothekselemente der DDB-Dateien des aktuellen Verzeichnisses erstellt werden. Die durch **libcrref.ulc** erzeugte Ausgabe enthält ein Listing aller selektierten Bibliothekselemente mit Angaben darüber, welche Symbole in welchen Dateien (ggf. mehrfach) definiert sind. Das Listing wird in einem Popupmenü mit Ausgabeoption angezeigt.

LISTDDB (STD) -- DDB-Dateielemente auflisten

Das Programm **listddb.ulc** listet die Datenbankelemente einer selektierbaren DDB-Datei in einem Popupmenü mit Ausgabeoption auf.

LOADELEM (STD) -- Laden Element mit Pruefung

Das Programm **loadelem.ulc** lädt ein menüselektierbares BAE-(Projekt-)Element. Bei Bedarf kann optional eine Konsistenzprüfung hinsichtlich fehlender bzw. falscher Bibliotheksdefinitionen oder noch nicht bzw. mit falschem Gehäuse platzierter Bauteile durchgeführt werden.

LOADFONT (STD) -- Laden Zeichensatz

Das Programm **loadfont.ulc** lädt einen menüselektierbaren Zeichensatz.

LOADNEXT (STD) -- Laden naechstes Element mit Pruefung

Das Programm **loadnext.ulc** ermittelt Name und Klasse des aktuell geladenen Elements und lädt automatisch das Element derselben Klasse, welches in der Namensliste der zugehörigen DDB-Datei direkt nach dem aktuell geladenen eingetragen ist (vorwärtsblättern). Ist das zu ladende Element auf diese Weise nicht zu ermitteln, dann wird ein menüselektierbares Element geladen. Bei Bedarf kann optional eine Konsistenzprüfung hinsichtlich fehlender bzw. falscher Bibliotheksdefinitionen oder noch nicht bzw. mit falschem Gehäuse platzierter Bauteile durchgeführt werden.

LOADPREV (STD) -- Laden vorheriges Element mit Pruefung

Das Programm **loadprev.ulc** ermittelt Name und Klasse des aktuell geladenen Elements und lädt automatisch das Element derselben Klasse, welches in der Namensliste der zugehörigen DDB-Datei direkt vor dem aktuell geladenen eingetragen ist (rückwärtsblättern). Ist das zu ladende Element auf diese Weise nicht zu ermitteln, dann wird ein menüselektierbares Element geladen. Bei Bedarf kann optional eine Konsistenzprüfung hinsichtlich fehlender bzw. falscher Bibliotheksdefinitionen oder noch nicht bzw. mit falschem Gehäuse platzierter Bauteile durchgeführt werden.

LROTATE (STD) -- Pickelement Linksdrehung, Winkelrichtung wechseln

Das Programm **lrotate.ulc** ermittelt die aktuell aktive BAE-Menüfunktion der aktuellen Interpreterumgebung und führt (sofern zulässig bzw. möglich) eine Untermenüfunktion zur Drehung des aktuell bearbeiteten Objekts um 90 Grad nach links durch; die Funktionsfähigkeit dieses Programms ist nur gewährleistet, wenn es implizit über Tastendruck (z.B. l bzw. L) aufrufbar ist.

Im **Layouteditor** kann mit der Funktion L/R Drehwinkel des **User Language**-Programms **GEDPART** der von **LROTATE** zu verwendende Drehwinkel für Bauteile, Pins, Polygone, Texte und Gruppen auf einen (beliebigen) von 90 Grad abweichenden Wert eingestellt werden.

MACRO (STD) -- Makrokommandoverwaltung

Das Programm **macro.ulc** bietet Funktionen zum Erstellen und Abrufen von Makrokommandosequenzen.

MIRROFF (STD) -- Pickelement Spiegelung aus

Das Programm **mirroff.ulc** ermittelt die aktuell aktive BAE-Menüfunktion der aktuellen Interpreterumgebung und führt (sofern zulässig bzw. möglich) eine Untermenüfunktion zum Zurücksetzen des Spiegelungsmodus des aktuell bearbeiteten Objekts durch; die Funktionsfähigkeit dieses Programms ist nur gewährleistet, wenn es implizit über Tastendruck (z.B. n bzw. N) aufrufbar ist.

MIRRON (STD) -- Pickelement Spiegelung ein, Editierrichtung wechseln

Das Programm **mirron.ulc** ermittelt die aktuell aktive BAE-Menüfunktion der aktuellen Interpreterumgebung und führt (sofern zulässig bzw. möglich) eine Untermenüfunktion zur Spiegelung des aktuell bearbeiteten Objekts durch; die Funktionsfähigkeit dieses Programms ist nur gewährleistet, wenn es implizit über Tastendruck (z.B. m bzw. M) aufrufbar ist.

MMB (STD) -- Interaktion Mittlere Maustaste

Das Programm **mmb.ulc** simuliert die Betätigung der mittleren Maustaste zur Aktivierung des Online-Bilddarstellungsmenüs. **MMB** ist für die Zuweisung auf eine Standardtaste (z.B. Leertaste) zur komfortablen Aktivierung des Menüs Bilddarstellung über die Tastatur vorgesehen.

MSMODE (STD) -- Maus Kontextmodus selektieren

Mit dem Programm **msmode.ulc** kann ein Mausoperationsmodus (Keine Operation, Kontextfunktionen, Löschen, Bewegen, Selektieren) selektiert werden, welcher objektspezifische Funktionen festlegt, die automatisch auf mit der linken Maustaste angeklickte Elemente angewendet werden.

OSSHELL (STD) -- Betriebssystem-Shell aktivieren

Das Programm **osshell.ulc** startet einen Kommandointerpeter zur Ausführung von Betriebssystemkommandos. Hierzu wird die **User Language**-Funktion system zum Aktivieren der Kommandos benutzt. Vor Benutzung dieses Programms sollten Sie unbedingt die Einschränkungen hinsichtlich der Anwendung der Funktion **system** beachten!

RENAMEEL (STD) -- DDB-Dateielemente umbenennen

Das Programm **renameel.ulc** bietet eine Funktion zur DDB-Dateielement-Umbenennung. Verfügbare Elementklassen werden hierbei in Abhängigkeit von der aktuellen Interpreterumgebung angeboten.

RROTATE (STD) -- Pickelement Rechtsdrehung, Rechteck Zeichnen

Das Programm **rrotate.ulc** ermittelt die aktuell aktive BAE-Menüfunktion der aktuellen Interpreterumgebung und führt (sofern zulässig bzw. möglich) eine Untermenüfunktion zur Drehung des aktuell bearbeiteten Objekts um 90 Grad nach rechts durch; die Funktionsfähigkeit dieses Programms ist nur gewährleistet, wenn es implizit über Tastendruck (z.B. r bzw. R) aufrufbar ist.

Im **Layouteditor** kann mit der Funktion L/R Drehwinkel des **User Language**-Programms **GEDPART** der von **RROTATE** zu verwendende Drehwinkel für Bauteile, Pins, Polygone, Texte und Gruppen auf einen (beliebigen) von 90 Grad abweichenden Wert eingestellt werden.

SAVEELAS (STD) -- Ablegen auf Namen

Das Programm **saveelas.ulc** speichert das aktuell geladene Element unter einem frei wählbaren Datei- und Elementnamen. Existiert das selektierte Zielelement bereits, dann erfolgt eine Abfrage, ob dieses Element überschrieben werden soll.

SIZE (STD) -- Größenänderung aktuelles Element

Das Programm **size.ulc** ermittelt die aktuell aktive BAE-Menüfunktion der aktuellen Interpreterumgebung und führt (sofern zulässig bzw. möglich) eine Untermenüfunktion zur Änderung der Göße des aktuell bearbeiteten Objekts durch; die Funktionsfähigkeit dieses Programms ist nur gewährleistet, wenn es implizit über Tastendruck (z.B. s bzw. S) aufrufbar ist. Die neue Objektgröße wird über ein Optionsmenü mit einer Liste vordefinierter Werte und einer Funktion zur Eingabe eines spezifischen Wertes eingestellt.

SMALLER (STD) -- Pickelement verkleinern/verschmälern

Das Programm **smaller.ulc** ermittelt die aktuell aktive BAE-Menüfunktion und führt (sofern zulässig bzw. möglich) eine Untermenüfunktion zur Verkleinerung bzw. Verschmälerung des aktuell bearbeiteten Objekts durch; die Funktionsfähigkeit dieses Programms ist nur gewährleistet, wenn es implizit über Tastendruck (z.B. B) aufrufbar ist.

STEPDOWN (STD) -- Um eine Lage runterwechseln

Das Programm **stepdown.ulc** ermittelt die aktuell aktive BAE-Menüfunktion und führt im **Layouteditor** (sofern zulässig bzw. möglich) eine Untermenüfunktion zum Wechsel auf die nächstniedrigere Lage aus. Die Funktionsfähigkeit dieses Programms ist nur gewährleistet, wenn es implizit über Tastendruck (z.B.], Ctrl/Mausrad abwärts) aufrufbar ist.

STEPUP (STD) -- Um eine Lage hochwechseln

Das Programm **stepup.ulc** ermittelt die aktuell aktive BAE-Menüfunktion und führt im **Layouteditor** (sofern zulässig bzw. möglich) eine Untermenüfunktion zum Wechsel auf die nächsthöhere Lage aus. Die Funktionsfähigkeit dieses Programms ist nur gewährleistet, wenn es implizit über Tastendruck (z.B. [, Ctrl/Mausrad aufwärts) aufrufbar ist.

TBATTACH (STD) -- Toolbar ausrichten

Mit dem Programm **tbattach.ulc** kann die mit **TOOLBAR** aktivierte Werkzeugleiste an einer der vier Begrenzungen (links, rechts, oben oder unten) des Arbeitsbereiches angeordnet werden. Wahlweise besteht auch die Möglichkeit, die Werkzeugliste auszublenden.

TOOLBAR (STD) -- Toolbar

Das Programm **toolbar.ulc** aktiviert eine Werkzeugleiste (Toolbar) mit Zusatzfunktionen zur grafikgestützten Verwaltung von Entwurfsansichten und Schaltflächen für den komfortablen Aufruf häufig benötigter Dateizugriffs-, Zoom- und Reportfunktionen und zur Einstellung des Mausoperationsmodus.

UIFDUMP (STD) -- Menu- und Tastaturbelegung ausgeben

Das Programm **uifdump.ulc** erzeugt ein Listing über die komplette Menü- und Tastaturbelegung des aktuell aktiven BAE-Programm-Moduls. Das Listing wird in einem Popupmenü mit Ausgabeoption angezeigt.

UIFRESET (STD) -- Menue- und Tastaturbelegung zuruecksetzen

Das Programm **uifreset.ulc** setzt die komplette Menü- und Tastaturbelegung des aktuell aktiven BAE-Programm-Moduls zurück.

UIFSETUP (STD) -- Menue- und Tastaturbelegung aktivieren

Das Programm **uifsetup.ulc** dient der automatisierten Aktivierung einer vordefinierten Menü- und Tastaturbelegung im aktuell aktiven BAE-Programm-Modul. **UIFSETUP** kann durch Aufruf über das **User Language**-Programm **BAE_ST** automatisiert über die Startupsequenz der einzelnen Interpreterumgebungen aktiviert werden.

ZOOMIN (STD) -- Zoom größer

Das Programm **zoomin.ulc** führt das Zoom größer-Kommando im **Bartels AutoEngineer** aus.

ZOOMOUT (STD) -- Zoom kleiner

Das Programm **zoomout.ulc** führt das Zoom kleiner-Kommando im **Bartels AutoEngineer** aus.

4.2.2 SCM-Programme

Die nachfolgend aufgeführten **User Language**-Programme sind in der Interpreterumgebung des **Schaltplaneditor** ablauffähig.

ATTRSET (SCM) -- SCM-Symbol Attributwertzuweisung

Das Programm **attrset.ulc** aktiviert einen Dialog mit erweiterten Funktionen zur Zuweisung von Attributwerten an mausselektierbare SCM-Symbole.

CHKSMAC (SCM) -- Nicht definierte SCM-Makroreferenzen auflisten

Das Programm **chksmac.ulc** listet die Namen aller undefinierten Makroreferenzen des aktuell geladenen SCM-Elements in einem Popupmenü mit Ausgabeoption auf.

DEF2CSV (SCM) -- Symboldatenbank aus .def-Dateien erzeugen

Das Programm **def2csv.ulc** durchsucht ein auswählbares Verzeichnis nach `.def`-Dateien mit logischen Bauteildefinitionen und erzeugt daraus `.csv`- und `.map`-Dateien zur Erzeugung von Symbolauswahldatenbanken mit den **User Language**-Programmen **symattdb.ulc** und **symmapdb.ulc**. Die Bibliotheksnamen für die Symbole werden aus dem Namen der `.def`-Datei abgeleitet. D.h., **def2csv.ulc** sollte nur auf `.def`-Dateien angewendet werden für eine Symboldatenbank mit einem entsprechenden Dateinamen existiert.

FINDSPRT (SCM) -- SCM-Bauteilsuche

Das Programm **findsprt.ulc** dient der Suche von Bauteilen im aktuell geladenen Stromlaufplan. Das Auswahlmenü zur Selektion des zu suchenden Bauteils zeigt sämtliche Bauteile des aktuellen Stromlaufplans (mit allen zugehörigen Blättern), wobei für jedes Bauteil der Planname, der SCM-Bauteilname und der vom **Packager** zugewiesene physikalische Bauteilname angegeben ist. Soll ein Bauteil auf einem anderem als dem aktuell geladenen Stromlaufblatt gesucht werden, dann erfolgt im Bedarfsfall eine Abfrage, ob das aktuell geladene Element gesichert bzw. die Suche tatsächlich durchgeführt werden soll. Zusatzoptionen ermöglichen eine Suche direkt nach SCM- bzw. Layoutbauteilname sowie die Lokalisierung von Bauteilen mit selektierbaren Attributwerten. Das Programm führt mit der Bilddarstellungsfunktion Fenster Mitte automatisch einen Zoom auf das gefundene Bauteil aus.

LOGLEDIT (SCM) -- Loglib-Editor-Funktionen

Mit dem Programm **logledit.ulc** können logische Bibliotheksdefinitionen im **Schaltplaneditor** geladen, editiert und kompiliert werden. Auf SCM-Symbolebene wird die logische Bibliotheksdefinition des aktuell bearbeiteten Symbols geladen. Es wird eine Templatedefinition mit allen Symbolpins automatisch erzeugt, sofern noch keine zugehörige logische Bibliotheksdefinition existiert. Auf Schaltplanebene kann die zu bearbeitende Loglibdefinition durch Selektion eines auf dem Stromlaufblatt platzierten Symbols gewählt werden. Bei allen anderen Elementklassen erfolgt eine Symbolnamensabfrage. Die zugehörige Loglibdefinition kann dann entweder aus der aktuell bearbeitenden Designdatei, der Defaultbibliothek oder einer beliebigen anderen DDB-Datei geladen werden. Sobald die Loglibdefinition geladen ist, kann sie editiert und abschließend mit dem Button OK kompiliert werden. Das Format der Loglibdefinitionen ist ausführlich in der Beschreibung des Utilityprogramms **LOGLIB** dokumentiert.

NETCONV (SCM) -- Logische Netzliste einlesen

Das Programm **netconv.ulc** dient dazu, logische (d.h. ungepackte) Netzlisten aus unterschiedlichen ASCII-Formaten (BAE, ALGOREX, Applicon, CADNETIX, CALAY, EEDESIGNER, Marconi ED, Mentor, MULTIWIRE, OrCAD, PCAD, RINF, SCICARDS, TANGO, VECTRON, VUTRAX, WIRELIST) in den **Bartels AutoEngineer** zu übertragen. Nach Auswahl der Netzlistendatei `<project>.net` werden die Netzlistendaten eingelesen und die darin enthaltene logische Netzliste wird in der DDB-Datei `<project>.ddb` unter dem in der Eingabedatei spezifizierten Namen (Default `netlist`) abgelegt. Diese logische Netzliste lässt sich dann mit dem **Packager** in eine physikalische (d.h. gepackte) Netzliste umwandeln. Anschließend kann das zugehörige Layout erstellt werden, wobei dann auch Pin- und Gattertausch entsprechend den durch den **Packager** übertragenen `swap`-Kommandos aus der logischen Bibliothek durchgeführt werden kann. Für Konsistenzprüfungen werden die Pinlisten der in der Netzliste referenzierten SCM-Symbole benötigt; **NETCONV** sucht zunächst in der Zielprojektdatei (`<project>.ddb`) und anschließend in einer im Quellcode vordefinierten Liste von SCM-Bibliotheksdateien nach den erforderlichen Symboldefinitionen. Netzattribute zur **Autorouter**-Steuerung (`ROUTWIDTH`, `POWWIDTH`, `MINDIST`, `PRIORITY`) werden durch den Anschluss synthetisch generierter Netzattribut-Bauteile übertragen; die hierfür benötigten Namensreferenzen (Netzattribut-Symbolname, Pinname, Netzattributname) sind im Quellcode vordefiniert. Der Fortgang der Bearbeitung sowie eventuell auftretende Fehler und Warnungen werden am Bildschirm und in einer Protokolldatei mit Namen `bae.log` aufgelistet. Nach erfolgreicher Übernahme der Netzliste erfolgt der Hinweis, dass ein **Packager**-Lauf durchzuführen ist.

PERRLIST (SCM) -- Anzeige Packager-Fehlerliste

Das Programm **perrlist.ulc** aktiviert ein permanent sichtbares Dialogfenster zur Anzeige von **Packager**-Fehlermeldungen mit Optionen zum Zoom auf selektierbare Fehlersymbole.

PLANSORT (SCM) -- SCM-Plan sortieren/nummerieren

Das Programm **plansort.ulc** führt für selektierbare DDB-Dateien des aktuellen Verzeichnisse eine automatische Benennung bzw. Nummerierung der SCM-Plannamen durch.

SADDNAME (SCM) -- SCM-Symbolname definieren

Das Programm **saddname.ulc** ermöglicht die halbautomatische Platzierung eines Textes dessen Inhalt dem Makronamen des aktuell geladenen SCM-Symbols entspricht.

SAUTONAM (SCM) -- Automatische Schaltplansymbolbenennung/-Nummerierung

Das Programm **sautonam.ulc** ermöglicht die Aktivierung verschiedener Funktionen zur automatischen Umbenennung bzw. Nummerierung von Bauteilen auf dem aktuell bearbeiteten Schaltplan. Die Nummerierung der Bauteile erfolgt jeweils von links oben nach rechts unten auf dem Stromlaufblatt. Die Bauteilnummerierung kann wahlweise auf das aktuell geladene Stromlaufblatt (Option Aktuelles Blatt) oder auf alle Blätter des aktuell geladenen Stromlaufplans angewendet werden (Option Alle Blätter).

SBROWSE (SCM) -- Schaltplansymbolbrowser

Das Programm **sbrowse.ulc** ermöglicht die Auswahl zu ladender bzw. zu platzierender SCM-Bibliothekselemente, wobei in einem Popupmenü eine grafische Voranzeige des aktuell selektierten Bibliothekselements erfolgt.

SCM_GRPL (SCM) -- SCM Gruppenladeaktion

Das Programm **scm_grpl.ulc** wird automatisch nach dem Laden von Gruppen aktiviert. **SCM_GRPL** aktualisiert Symbolvariantenattribute.

SCM_MS (SCM) -- SCM Mausaktion

Das Programm **scm_ms.ulc** wird automatisch beim Drücken der linken Maustaste gestartet wenn gerade keine andere **Schaltplaneditor**-Funktion aktiv ist. **SCM_MS** aktiviert ein kontextsensitives Funktionsmenü für das Objekt an der aktuellen Mausposition. Ist kein Element geladen, dann werden die Funktionen Element Laden, Neues Element und Mausmodus angeboten.

SCM_MSG (SCM/HighEnd) -- SCM Messagesystem-Aktion

Das Programm **scm_msg.ulc** empfängt und bearbeitet Messages aus anderen **BAE HighEnd** Modulen. Die auszulösenden Aktionen (Setzen von Variablen, Aufruf von **User Language**-Programmen) sowie die zu bearbeitenden Objekte werden jeweils durch die eingehende Meldung bestimmt.

SCM_PLC (SCM) -- SCM Symbolplatzierungsaktion

Das Programm **scm_plc.ulc** wird automatisch nach der Platzierung eines Symbols aktiviert. **SCM_PLC** aktualisiert den Platzierungssymbolpool ("Warenkorb").

SCMBOUND (SCM) -- SCM-Arbeitsbereich/Elementgrenzen setzen

Das Programm **scmbound.ulc** führt wahlweise eine Vergrößerung oder Verkleinerung der Elementgrenzen des aktuell geladenen Elements durch.

SCMCON (SCM) -- SCM-Verbindungsfunktionen

Das Programm **scmcon.ulc** ermöglicht die Aktivierung spezieller Verbindungsfunktionen wie Änderung des Busdarstellungsmodus, Busabfrage oder Netzhighlight.

SCMCRREF (SCM) -- SCM-Plan Bauteil-/Labelliste erstellen

Das Programm **scmcrref.ulc** erzeugt eine Bauteil- und Label-Cross-Referenz für den Stromlauf einer selektierbaren Projektdatei und zeigt diese in einem Popupmenü mit Ausgabeoption an. Datei aus.

SCMDISP (SCM) -- SCM-Bilddarstellungsfunktionen

Das Programm **scmdisp.ulc** ermöglicht die Aktivierung einer Reihe zusätzlicher Bilddarstellungsfunktionen im **Schaltplaneditor**.

SCMDRAW (SCM) -- SCM-2D-Zeichnungsfunktionen

Das Programm **scmdraw.ulc** ermöglicht die Aktivierung verschiedener 2D-Zeichnungsfunktionen zur Generierung von Kreisen, Rechtecken und Pfeilen, zur Distanzbemaßung sowie zur Linealerzeugung.

SCMDUMP (SCM) -- SCM-ASCII-Dump

Das Programm **scmdump.ulc** gibt einen ASCII-Dump des aktuell geladenen SCM-Elements auf Datei aus.

SCMDXFDI (SCM) -- SCM AutoCAD/DXF-Übernahme

Das Programm **scmdxfdi.ulc** ermöglicht das Einlesen von Zeichnungsdaten im Format AUTOCAD-DXF auf das aktuell geladene Schaltplanelement, wobei Eingabe-Längeneinheiten, Einleseoffset und zu übernehmende Lagen frei wählbar sind.

SCMDXFDO (SCM) -- SCM AutoCAD-DXF-Ausgabe

Das Programm **scmdxfdo.ulc** erzeugt eine AUTOCAD-DXF-Ausgabe für das aktuell geladene SCM-Element.

SCMEDFDI (SCM) -- SCM EDIF-Daten importieren

Das Programm **scmedfdi.ulc** dient dazu, Schaltplandaten, d.h. Netzlisten, Schaltzeichen und Stromlaufpläne aus dem Format EDIF in den **Schaltplaneditor** des **Bartels AutoEngineer** zu übertragen. Die für den anschließenden **Packager**-Lauf erforderlichen Definitionen in der logische Bauteilbibliothek werden nach Bedarf automatisch erzeugt.

SCMEPS (SCM) -- SCM EPS/PDF Ausgabe

Das Programm **scmeps.ulc** erzeugt für das aktuell geladene SCM-Element wahlweise eine Ausgabe im Adobe Portable Document Format (PDF) oder im Encapsulated PostScript (EPS) Format, wobei verschiedene Skalierungsfaktoren für die Ausgabe zur Auswahl stehen. Über eine Option kann ein interaktiv festlegbarer Bereich für die Ausgabe selektiert werden. Es werden die verschiedenen Grafikobjekte (Standard- bzw. Kommentartext, Dokumentarflächen, Dokumentarlinien, gestrichelte Linien, Kontaktbereiche, Verbindungen, Busse) der unterschiedlichen Hierarchieebenen (Stromlaufblatt, Symbol, Label, Marker) simultan geplottet, wobei zusätzlich Ausgabeoptionen für Füllmodus, Farb- bzw. Grauwert, Schraffur und gestrichelte Linien spezifiziert werden können. Die auszugebenden Objekte sowie die Ausgabeoptionen sind in einer - an firmenspezifische Konventionen anpaßbaren - speziellen Variablen im Quellcode von **SCMEPS** festgelegt. Darüberhinaus werden mit **SCMRULE** an Plotelemente zugewiesene Plotsichtbarkeitsregeln entsprechend berücksichtigt.

SCMGROUP (SCM) -- SCM-Gruppenfunktionen

Das Programm **scmgroup.ulc** ermöglicht die automatische Selektion bzw. De-Selektion aller Objekte eines selektierbaren Typs oder mit spezifischen Attributen, das Laden von Gruppen aus anderen Datenbankhierarchieebenen, die Änderung der Größe gruppenselektierter Texte, sowie die globale Attributwertzuweisung an gruppenselektierte Bauteile. Eine Spezialfunktion zur Selektion von Layoutbauteilmengen entsprechend der selektierten SCM-Symbole wird ebenfalls angeboten.

SCMIO (SCM) -- SCM-Ein-/Ausgabefunktionen

Das Programm **scmio.ulc** aktiviert im **Schaltplaneditor** ein Menü mit verschiedenen SCM-spezifischen Ein-/Ausgabefunktionen. Über das Kommando **`addioitem`** können in der Datei **`bae.ini`** im BAE-Programmverzeichnis in einfacher Weise zusätzliche (benutzerspezifische) Import-/Exportfunktionen in das Menü aufgenommen werden.

SCMMACL (SCM) -- Schematic Makro Laden

Das Programm **scmmacl.ulc** l&aeuml;dt das Bibliothekselement eines mausselektierbaren Symbols, Labels oder Markers des aktuell geladenen SCM-Elements in den **Schaltplaneditor**.

SCMPART (SCM) -- SCM-Symbol-/Labelfunktionen

Das Programm **scmpart.ulc** ermöglicht die Aktivierung einer Reihe von Symbol- und Labelfunktion wie z.B. Drehen bzw. Rotieren selektierbarer Symbole bzw. Labels, Bauteilumbenennung, Symbol- und Labelabfrage, Bauteilsuche, Stücklistenausgabe, menügesteuerte Attributwertzuweisung, usw.

SCMPCR (SCM) -- SCM-Report

Das Programm **scmpcr.ulc** zeigt einen Report über das aktuell geladene SCM-Element in einem Popupmenü mit Ausgabeoption an.

SCMPEDIT (SCM) -- SCM Positionspick/Elementdatenmanipulation

Das Programm **scmpedit.ulc** bietet Funktionen zur automatischen Positions- bzw. Pickpunktanwahl anhand der an der aktuellen Mausposition platzierten Elemente. Die Funktionsfähigkeit dieses Programms ist nur gewährleistet, wenn es implizit über Tastendruck (z.B. p oder P) aufrufbar ist.

SCMPLOT (SCM) -- SCM-Plan plotten

Das Programm **scmplot.ulc** plottet unter Verwendung der aktuell eingestellten Plotparameter (für Ausgabekanal, Maßstab, etc.) sämtliche SCM-Pläne einer selektierbaren DDB-Datei, wobei zwischen den Ausgabeformaten PostScript und HP-Laser (PCL) gewählt werden kann.

SCMPOLY (SCM) -- SCM-Polygonfunktionen

Das Programm **scmpoly.ulc** ermöglicht die Aktivierung einer Reihe von Polygonbearbeitungsfunktionen wie z.B. die Definition bzw. Änderung von Polygontypen, das Spiegeln von Polygonen, das Kopieren gruppenselektierter Polygone mit Skalierung, die Aktivierung verschiedener 2D-Zeichnungsfunktionen, usw.

SCMRULE (SCM) -- SCM-Regelzuweisungsutility

Mit dem Programm **scmrule.ulc** können im **Schaltplaneditor** Regeln an Schaltplanelemente zugewiesen werden. Um Regelsystemfehler zu vermeiden, sind die zugewiesenen Regeln mit dem Regelsystem-Compiler **RULECOMP** zu definieren bzw. zu kompilieren.

SCMSETUP (SCM) -- Schematic Editor Setup

Das Programm **scmsetup.ulc** stellt eine Reihe von SCM-Parametern und Bilddarstellungsmodi auf Defaultwerte ein.

SCMTEXT (SCM) -- SCM-Textfunktionen

Das Programm **scmtext.ulc** ermöglicht die Aktivierung einer Reihe von Textfunktionen wie z.B. Ändern und Ersetzen von Texten, Setzen von Textgrößen, Umwandlung selektierbarer Texte in Standard- oder Kommentartext, Zuweisung von Textklassen zur Steuerung der Textsichtbarkeit, Konvertierung von Texten in Flächen oder Linien, Generierung kreisbogenförmig geschriebener Texte, usw.

SCMVAR (SCM) -- Schaltplanvariante selektieren

Das Programm **scmvar.ulc** aktiviert eine selektierbare Schaltplanvariante mit optionalem Transfer variantenspezifischer Daten.

SLABCHK (SCM) -- SCM Labelnamen ueberpruefen

Das Programm **slabchk.ulc** prüft die Labels sämtlicher Stromlaufblätter eines Projekts durch. Labels bzw. Netznamen, die nur einmal im gesamten Schaltplan definiert bzw. referenziert sind, werden als mögliche Fehler gekennzeichnet.

SLIBCOMP (SCM) -- SCM-Bibliothekselemente vergleichen

Das Programm **slibcomp.ulc** ist ein Utilityprogramm für das SCM-Bibliotheksmanagement. **slibcomp.ulc** vergleicht die Bibliotheksdefinitionen (Pinnamen, Pinplatzierung, Verwendung von Pinmakros, Zuordnung zu Einträgen in der logischen Bibliothek, etc.) selektierbarer Stromlaufsymbole und zeigt das Ergebnis des Bibliotheksvergleichs in einem Popupmenü mit Ausgabeoption an.

SLIBDOC (SCM) -- SCM-Bibliotheksdokumentation

Das Programm **slibdoc.ulc** erzeugt in selektierbaren SCM-Bibliotheksdateien des aktuellen Verzeichnisses Stromlaufblätter, auf die sortiert nach Namen die in der jeweiligen DDB-Datei enthaltenen SCM-Symbole automatisch platziert werden. Die so erzeugten Stromlaufblätter können optional im Batchbetrieb im Format PostScript ausgeplottet und als Bibliotheks-Dokumentation (Bauteilmappe) verwendet werden. Die Dokumentation kann wahlweise für SCM-Symbole oder SCM-Labels erstellt werden, wobei eine Selektion über Symbolnamensmuster möglich ist. Es stehen unterschiedliche Blattformate für die Ausgabe zur Verfügung.

SLIBNEWS (SCM) -- SCM-Bibliothek mit neuen Symbolen erzeugen

Das Programm **slibnews.ulc** ist ein Utilityprogramm für das SCM-Bibliotheksmanagement. Mit **SLIBNEWS** werden aus einer selektierbaren DDB-Datei die SCM-Elemente, die nicht in der aktuellen BAE-Bibliothek enthalten sind, in eine Temporärdatei kopiert. Aus der so erzeugten Bibliotheksdatei kann anschließend eine kontrollierte Übernahme bzw. Freigabe der neuen SCM-Bibliothekselemente erfolgen.

SLIBUTIL (SCM) -- SCM-Bibliotheksmanagement

Das Programm **slibutil.ulc** ermöglicht die Aktivierung einer Reihe von SCM-Bibliotheksmanagement-Utilities wie etwa zur Erzeugung von Bibliotheks-Cross-Referenzen, zur automatischen Generierung von Bibliotheks-Dokumentation, zum Kopieren bzw. Löschen menüselektierbarer DDB-Dateielemente, zur Durchführung von Bibliotheks-Konsistenzprüfungen, zum Exportieren von Loglib-Definitionsdateien, usw.

SMOVPINN (SCM) -- SCM-Symbol-Pinnamen verschieben

Das Programm **smovpinn.ulc** bewegt die Namen der oberen und unteren Pins des aktuell geladenen SCM-Symbols auf die jeweiligen Pin-Nullpunkte (und gibt dadurch bei geeigneter Pin-Makro-Definition die Bereiche für das Verlegen von Netzanschlüssen an diese Pins frei).

SNEXTSYM (SCM) -- SCM Folgesymbol platzieren

Das Programm **snextsym.ulc** platziert das nächste Symbol aus dem Projektsymbolpool ("Warenkorb") bzw. wiederholt die Platzierung des vorherigen Symbols, falls der Pool leer ist.

SPICESIM (SCM) -- Spice-Netzlistenausgabe

Das Programm **spicesim.ulc** dient der Generierung von Spice-Netzlisten. Es können wahlweise Netzlisten für ein ganzes Schaltplanblatt (Option Gesamtes Blatt) oder für die in der Gruppe selektierten Symbole (Option Gruppensymbole) ausgegeben werden. Die Ausgabe erfolgt auf eine Datei mit der Dateiendung `.cir`. In diese Datei sollten zusätzliche Kommandos wie etwa `.LIB` und `.INCLUDE` zur Adressierung der Spice-Bibliotheken und zur Definition von Kontrollparametern für die Spice-Simulation eingetragen werden. Die Zuweisung von Spice-Modelltypen an SCM-Symbole und die Festlegung der Ausgabereihenfolge der Symbolpins kann mit Hilfe der Funktionen Spice Modell und Spice Pinabfolge des **User Language**-Programms **SCMRULE** vorgenommen werden.

SPOPCOL (SCM) -- SCM-Farbauswahl

Das Programm **spopcol.ulc** aktiviert ein Popupmenü zur Anzeige und Definition der aktuellen Farbeinstellungen im **Schaltplaneditor**.

SSPINMAC (SCM) -- SCM-Symbol Pinmakros setzen

Das Programm **sspinmac.ulc** setzt alle Marker-Makros des aktuell geladenen SCM-Elements automatisch auf ihren Default zurück.

SSVGOUT (SCM) -- Schaltplan SVG (Scalable Vector Graphics) Ausgabe

Das Programm **ssvgout.ulc** generiert aus dem aktuell geladenen Schaltplanelement eine Exportdatei im Format SVG (Scalable Vector Graphics).

SSYMATTR (SCM) -- SCM-Symbol Attributdatenbankmanagement

Das Programm **ssymattr.ulc** enthält SQL-Routinen zur Definition und Pflege vordefinierter Bauteilattribute für Kondensator- und Widerstandsreihen in einer relationalen Datenbank. Darüber hinaus stellt **ssymattr.ulc** auf Schaltplanebene entsprechende Funktionen zur halbautomatischen Bauteil-Attributdefinition und zum Setzen von Defaultattributen bereit.

SSYMEDIT (SCM) -- SCM-Symbol-Editierfunktionen

Das Programm **ssymedit.ulc** ermöglicht bei geladenem SCM-Symbol den menügesteuerten Aufruf einer Reihe von Symbol-Editierfunktionen. Zur Auswahl stehen unter anderem Routinen zum Setzen des Symbol-Nullpunkts, zum Platzieren von Pins, zum Umplatzieren von Pingruppen, zur Änderung der Elementgrenzen, usw.

SSYMORIG (SCM) -- SCM-Symbol-Nullpunkt setzen

Das Programm **ssymorig.ulc** setzt den Nullpunkt des aktuell geladenen SCM-Symbols automatisch auf einen selektierbaren Pin.

SSYMPATT (SCM) -- SCM-Symbol-Namensmustereinstellungen

Das Programm **ssympatt.ulc** zum Setzen von Basis-Namensmustern für die automatische Benennung von Symbolen. Auf Symbolebene kann ein Namensmuster spezifiziert werden. Auf Schaltplanebene kann ein Benennungmodus (Standard oder Nummernscan) und optional eine Startnummer für die Benennung angegeben werden.

STXFIN (SCM) -- TXF Schaltplan-Datenübernahme

Das Programm **stxfin.ulc** importiert TXF-Schaltplandaten aus einer selektierbaren ASCII-Datei in den **Schaltplaneditor**

SYMATTDB (SCM) -- SQL-Attributdatenbank zur SCM-Symbolselektion aufbauen

Das Programm **symattdb.ulc** gestattet die Generierung einer SQL-Datenbank zur Symbolselektion. Die Datenbank wird durch Importieren von `.csv`-Dateiinhalten aus einem selektierbaren Verzeichnis erzeugt.

SYMEDBAT (SCM) -- SCM-Symbol Batcheditor

Das Programm **symedbat.ulc** ermöglicht die Definition einer Reihe von Aktionen zum Editieren von Symbolen sowie die automatische Durchführung all dieser Editieraktionen für die SCM-Symbole selektierbarer SCM-Bibliotheksdateien des aktuellen Verzeichnisses. Zur Auswahl stehen Funktionen zum Definieren, Ändern oder Löschen spezieller Texte, zur Minimierung der Elementgrenzen, zum Rücksetzen von Pin-Makros, zum Setzen des Symbol-Nullpunkts, zur Zuweisung von Defaultattributen, zur automatisierten Platzierung von Texten, usw.

SYMMAPDB (SCM) -- SQL-Symbolgatterzuordnungsdatenbank aufbauen

Das Programm **symmapdb.ulc** gestattet die Generierung einer SQL-Datenbank zur Symbolgatterauswahl. Die Datenbankeinträge können aus einer `.map`-Datei importiert werden.

SYMSEL (SCM) -- SCM-Symbolplatzierung mit Auswahl aus Attributdatenbank

Das Programm **symsel.ulc** gestattet die Platzierung von Symbolen mit Auswahl aus einer mit **SYMATTDB** erzeugten Attribut- bzw. Symbolselektionsdatenbank.

TBDVSCM (SCM) -- SCM Entwurfsansichten in Toolbar verwalten

Das Programm **tbdvscm.ulc** bearbeitet Anforderungen zur Aktualisierung von SCM-Entwursfansichten und verwaltet ein Utility zur automatischen Zuweisung von Attributen an Bauteile und zur Platzierung von Symbolkopien.

4.2.3 Layout-Programme

Die nachfolgend aufgeführten **User Language**-Programme sind in den Interpreterumgebungen des **Layouteditor**, des **Autorouters** und des **CAM-Prozessors** ablauffähig.

AIRLDENS (LAY) -- Airline-Dichteverteilungsdiagramm

Das Programm **airldens.ulc** generiert eine farblich abgestufte grafische Anzeige der Dichteverteilung der Airlines des aktuell geladenen Layouts. Die Skalierung des Anzeigebereichs ergibt sich hierbei aus der Auflösung des aktuell eingesetzten Grafiksystems.

CHECKLNL (LAY) -- Layout gegen Netzliste pruefen

Das Programm **checklnl.ulc** prüft das aktuelle Layout gegen die Netzlistendaten und zeigt einen Report über noch nicht platzierte Bauteile, falsche Gehäusebauformen und fehlende Pindefinitionen in einem Popupmenü mit Ausgabeoption an.

CHKLMAC (LAY) -- Nicht definierte Layout-Makroreferenzen auflisten

Das Programm **chklmac.ulc** listet die Namen aller undefinierten Makroreferenzen des aktuell geladenen Layoutelements in einem Popupmenü mit Ausgabeoption auf.

CONBAE (LAY) -- Netzlistenausgabe

Das Programm **conbae.ulc** gibt die Netzliste des aktuell geladenen Layouts im BAE-ASCII-Netzlistenformat auf Datei aus, wobei wahlweise eine Sortierung der Bauteil- und Netznamen durchgeführt wird. Die Ausgabe von Bauteilplatzierungsdaten ebenso wie die Ausgabe von Netzen mit nur einem Pin und die Ausgabe automatisch generierter Testpunkte kann optional veranlasst werden.

DRILLOUT (LAY) -- Bohrdatenausgabe

Das Programm **drillout.ulc** gibt die Bohrdaten des aktuell geladenen Layouts im Sieb&Meier- oder im Excellon-Format aus. Daneben wird auch eine Option zur Anzeige einer Bohrdatenstatistik angeboten. Die Bohrklasse für die Ausgabe ist über Menü selektierbar. Die Dateinamen der Ausgabedateien werden automatisch generiert. Die Werkzeugtabelle wird selbsttätig ermittelt, sofern für die Bohrdatenausgabe nicht die im Programm vordefinierte Bohrertabelle verwendet wird. Ein Sortieralgorithmus zur Minimierung des Bohrverfahrwegs kann optional aktiviert werden. Durch einfache Änderung einer Compiler-Direktive kann das Programm so übersetzt werden, dass es bei der Bohrdatenausgabe eine grafische Anzeige des Bohrverfahrwegs generiert (Bohrverfahrweg-Simulation).

DUMPPLC (LAY) -- Platzierungsdatenausgabe

Das Programm **dumpplc.ulc** gibt die Platzierungsdaten des aktuell geladenen Layouts auf Datei aus, wobei zwischen generischem Format und BAE-Format gewählt werden kann. Bei der Ausgabe im BAE-Format kann wahlweise eine Rundung der Platzierungskoordinaten auf das aktuell eingestellte X-Eingaberaster veranlasst werden.

EDF20CON (LAY) -- EDIF 2.0 Netzliste importieren

Das Programm **edf20con.ulc** dient der Übernahme von EDIF-Netzlisten in den **Bartels AutoEngineer**. **EDF20CON** liest die selektierte EDIF-Datei mit der Dateinamensendung `.edn` und speichert die eingelesene Netzliste in einer DDB-Datei unter dem in der Eingabedatei für die Netzliste angegebenen Elementnamen (der über LAYDEFELEMENT mit dem Utilityprogramm **BSETUP** gesetzte Name wird verwendet, wenn kein Elementname in der Eingabedatei spezifiert ist). Für Konsistenzprüfungen wird der Zugriff auf die Pinlisten der in der Netzliste verwendeten Bauteilsymbole benötigt. Die entsprechenden Layoutsymbolinformationen werden entweder aus der Zielprojektdatei oder aus der aktuell selektierten Layoutbibliothek geladen. Erforderliche Bauteile, die noch nicht in der Projektdatei existieren, werden automatisch aus der Layoutbibliothek übernommen. **EDF20CON** generiert automatisch die für den nachfolgenden **Packager**-Lauf benötigten logischen Bibliotheksdefinitionen (1:1-Zuordnung der Bauteilgehäusezuweisungen). Während der Netzlistenumsetzung werden Statushinweise und ggf. Warnungen und Fehlermeldungen auf dem Bildschirm ausgegeben und zusätzlich in die Logdatei `bae.log` geschrieben. Nach erfolgreicher Übernahme der Netzliste gibt **EDF20CON** den Hinweis, dass vor der Weiterbearbeitung im Layout ein **Packager**-Lauf zur Umsetzung der eingelesenen logischen Netzliste notwendig ist.

Warnung

EDF20CON ist nur für EDIF-Netzlisten aus ORCAD getestet. Zur Übernahme von EDIF-Netzlisten aus anderen Schaltplanpaketen ist **EDF20CON** möglicherweise entsprechend anzupassen.

GENCAD (LAY) -- GENCAD 1.4 Layout Datenausgabe

Das Programm **gencad.ulc** erzeugt aus dem aktuell geladenen Layout ein Datenfile im Format GENCAD1.4 DAT.

HYPLYNX (LAY) -- HyperLynx Layout Simulationsdatenausgabe

Das Programm **hyplynx.ulc** erzeugt aus dem aktuell geladenen Layout eine HyperLynx (**`.HYP`**) Simulationsdatendatei. Der Lagenaufbau wird aus einer selektierbaren externen Datei mit der Dateinamenserweiterung **`.stk`** entnommen. Unregelmäß geformte Lötaugen werden in umschreibende Rechtecke umgewandelt. Die Signallage 1 wird als **`BOTTOM`** ausgegeben. Die aktuell als oberste Lage definierte Signallage wird als **`TOP`** ausgegeben. Innenlagen (Lage 2 bis Oberste Lage - 1) werden als **`Inner_Layer_n`** (mit **`n`** im Bereich von 2 bis Oberste Lage - 1) ausgegeben.

ICAPNET (LAY) -- ICAP-Netzlistenimport

Das Programm **icapnet.ulc** importiert gepackte (d.h. physikalische) Netzlisten aus dem Schaltplan-/Simulationstool ICAP/4 (Version 8.2.10 / 1843 oder neuer; Tango-Netzlisten-Export) der Firma Intusoft. **ICAPNET** liest die Netzlistendaten aus der selektierten Netzlistendatei **`project.net`** und speichert die Netzliste in der DDB-Datei **`project.ddb`** unter dem in der Netzlistendatei angegebenen Elementnamen für die Netzliste (default **`netlist`**) ab. **ICAPNET** generiert 1:1-Pinzuweisungen für Bauteile ohne logische Bibliotheksdefinitionen in der Standard-Layoutbibliothek. Hinweise, Warnungen und Fehlermeldungen über den Fortgang der Netzlistenumsetzung werden auf dem Bildschirm und in der Logdatei **`bae.log`** aufgelistet. Nach erfolgreicher Übernahme der logischen Netzliste startet **ICAPNET** automatisch den **Packager** welcher seinerseits die logische Netzliste in eine physikalische Netzliste mit Pin- und Gattertauschinformation entsprechend der logischen Bibliotheksdefinitionen für die weitere Bearbeitung im Layout umwandelt.

IPCOUT (LAY) -- IPC-D-356 Testdatenausgabe

Das Programm **ipcout.ulc** generiert für das aktuell geladene Layout eine Testdatenausgabe im Format IPC-D-356. Die Ausgabe erfolgt auf eine Datei. Durchkontaktierungen werden als Zentrumspunkte behandelt. Lötmaskendaten werden aus der Dokumentarlage 2 extrahiert. Die Ausgabe wird nach Netzen bzw. Bohrungen sortiert und enthält bohrklassenspezifische Kontaktierungsvorgaben. Am Ende der Ausgabedatei wird eine Zählstatistik für die PIns, Durchkontaktierungen und Bohrungen ausgegeben.

LAYDUMP (LAY) -- Layout-ASCII-Dump Ein-/Ausgabe

Das Programm **laydump.ulc** bietet Funktionen zur Eingabe und Ausgabe von Layoutelementdaten in einem generischen, für BNF-Parser geeigneten ASCII-Format an. Die Ausgabefunktion gibt einen ASCII-Dump des aktuell geladenen Layoutelements auf Datei aus, wobei unterschiedliche Längeneinheiten für die Ausgabe zur Auswahl stehen.

Hinweis

Die Ausgabefunktion ist in allen Layoutmodulen, die Eingabefunktion jedoch nur im **Layouteditor** verfügbar.

LAYDXFDO (LAY) -- Layout AutoCAD/DXF-Ausgabe

Das Programm **laydxfdo.ulc** erzeugt eine AUTOCAD-DXF-Ausgabe für das aktuell geladene Layoutelement. Es steht ein Menü zur Selektion der auszugebenden lagen zur Verfügung. Alternativ können auch die aktuell sichtbaren oder programmintern definierte lagen ausgegeben werden.

LAYEPS (LAY) -- Layout EPS/PDF Ausgabe

Das Programm **layeps.ulc** erzeugt für das aktuell geladene Layoutelement wahlweise eine Ausgabe im Adobe Portable Document Format (PDF) oder im Encapsulated PostScript (EPS) Format, wobei verschiedene Ausgabeformate und Skalierungsmodi zur Auswahl stehen. Es werden mehrere Lagen simultan geplottet, wobei für jede Lage zusätzlich Ausgabeoptionen für Füllmodus, Farb- bzw. Grauwert, Schraffur, gestrichelte Linien und Linienstärken spezifiziert werden können. Die auszugebenden Lagen sowie die Ausgabeoptionen sind in einer - an firmenspezifische Konventionen anpaßbaren - speziellen Variablen im Quellcode von **LAYEPS** festgelegt. Wahlweise können auch menüselektierbare Lagen oder die aktuell sichtbaren Lagen mit Farbauswahl bzw. Farbzuweisung oder automatischer Grauwertskalierung der aktuellen Farben geplottet werden. Das Programm unterstützt die aus dem **CAM-Prozessor** bekannten Spiegelungsoptionen. Über eine weitere Option kann ein interaktiv festlegbarer Bereich für die Ausgabe selektiert werden.

LAYERUSE (LAY) -- Layoutbibliothek Lagenbelegungsreport

Das Programm **layeruse.ulc** ist ein Utilityprogramm für das Layout-Bibliotheksmanagement. **layeruse.ulc** analysiert die Lagenbelegung einer selektierbaren DDB-Datei und gibt das Ergebnis dieser Analyse in Form eines Reports auf eine ASCII-Datei aus.

LAYPCR (LAY) -- Layout-Report

Das Programm **laypcr.ulc** zeigt einen Report über das aktuell geladene Layoutelement in einem Popupmenü mit Ausgabeoption an.

LAYZMBRD (LAY) -- Zoom auf Layout-Platinenumrandung

Das Programm **layzmbrd.ulc** führt einen Zoom auf die Layout-Platinenumrandung aus.

LBROWSE (LAY) -- Layoutsymbolbrowser

Das Programm **lbrowse.ulc** ermöglicht die Auswahl zu ladender bzw. zu platzierender Layoutbibliothekselemente, wobei in einem Popupmenü eine grafische Voranzeige des aktuell selektierten Bibliothekselements erfolgt.

LCIFOUT (LAY) -- Layout CIF-Datenausgabe

Das Programm **lcifout.ulc** generiert Caltech-CIF-Daten aus dem aktuell geladenen Layoutelement. Über einen Dialog können der Name der Ausgabedatei spezifiziert, alle oder nur gruppenselektierte Elemente zur Ausgabe ausgewählt und der CIF-Ausgabemodus (flach oder hierarchisch) selektiert werden.

LDEFMANG (LAY) -- Platzierungsvorgaben fuer Layoutbauteilmakros definieren

Das Programm **ldefmang.ulc** ermöglicht die Zuweisung von Platzierungsvorgaben (z.B. für Drehung und Spiegelung) aus dem Neuronalen Regelsystem an das aktuell geladenen Layout-Bibliothekselement. Diese Vorgaben werden von den entsprechenden BAE-Platzierungsfunktionen automatisch berücksichtigt. Um undefinierte Regelsystemfehler zu vermeiden, sollten Regeln mit den Bezeichnungen `rot0`, `rot90`, `rot180`, `rot270`, `mirroroff` und `mirroron` mit Hilfe des Regelsystemcompilers **RULECOMP** definiert sein (die mitgelieferte Quellcodedatei **partplc.rul** enthält die benötigten Regeldefinitionen).

LLIBCOMP (LAY) -- Layout-Bibliothekselemente vergleichen

Das Programm **llibcomp.ulc** ist ein Utilityprogramm für das Layout-Bibliotheksmanagement. **llibcomp.ulc** vergleicht die Bibliotheksdefinitionen (Pinnamen, Pinplatzierung, Verwendung von Padstackmakros, Zuordnung zu Einträgen in der logischen Bibliothek, etc.) selektierbarer Bauteilsymbole und zeigt das Ergebnis des Bibliotheksvergleichs in einem Popupmenü mit Ausgabeoption an.

LLIBNEWS (LAY) -- Layout-Bibliothek mit neuen Symbolen erzeugen

Das Programm **llibnews.ulc** ist ein Utilityprogramm für das Layout-Bibliotheksmanagement. Mit **llibnews.ulc** werden aus einer selektierbaren DDB-Datei die Layoutelemente, die nicht in der aktuellen BAE-Bibliothek enthalten sind, in eine Temporärdatei kopiert. Aus der so erzeugten Bibliotheksdatei kann anschließend eine kontrollierte Übernahme bzw. Freigabe der neuen Layout-Bibliothekselemente erfolgen.

LLIBUTIL (LAY) -- Layout-Bibliotheksmanagement

Das Programm **llibutil.ulc** ermöglicht die Aktivierung einer Reihe von Layout-Bibliotheksmanagement-Utilities wie etwa zur Erzeugung von Bibliotheks-Cross-Referenzen, zur automatischen Generierung von Bibliotheks-Dokumentation, zur Durchführung von Bibliotheks-Konsistenzprüfungen, zum Kopieren bzw. Löschen menüselektierbarer DDB-Dateielemente, oder zur automatisierten Änderung von Bibliotheksdefinitionen (Änderung von Lagenzuweisungen, Austausch von Makros, etc.).

LMACCREF (LAY) -- Layout-Makro-Querverweisliste erzeugen

Das Programm **lmaccref.ulc** ist ein Utilityprogramm für das Layout-Bibliotheksmanagement. **lmaccref.ulc** erzeugt ein Cross-Referenz-Listing für Layoutelemente einer selektierbaren DDB-Datei. Der so erzeugte Report wird in einem Popupmenü mit Ausgabeoption angezeigt und enthält Angaben darüber, welche Layout-Bibliothekselemente auf welchen Layoutelementen der jeweils nächsthöheren Hierarchieebene benutzt werden.

LSVGOUT (LAY) -- Layout SVG (Scalable Vector Graphics) Ausgabe

Das Programm **lsvgout.ulc** generiert aus dem aktuell geladenen Layoutelement eine Exportdatei im Format SVG (Scalable Vector Graphics).

NETSTAT (LAY) -- Datenbank mit Netzhighlight/-sichtbarkeit verwalten

Das Programm **netstat.ulc** bietet SQL-Datenbankfunktionen zum Speichern und Laden von Informationen bzw. Einstellungen zu Netzhighlights und Airlineanzeigen.

PARTLIST (LAY) -- Stücklistenausgabe

Das Programm **partlist.ulc** gibt eine Stückliste in unterschiedlichen Formaten (BAE oder CSV/DBF) für das aktuell geladene Layout aus. Die Ausgabe wird gegliedert nach Bauteiltypkategorien die durch eine im Quellcode definierte Liste von Bauteilnamensmustern (z.B. `r*` für Widerstände, `c*` für Kondensatoren, etc.) festgelegt wird. Eine weitere vordefinierte Liste von Attributnamen legt die Attributwerte fest, nach denen eine weitere Unterscheidung bzw. Sortierung der Bauteiltypen vorgenommen wird (z.B. `$val` zur gesammelten Auflistung aller Widerstände oder Kondensatoren mit demselben Wert, `$llname` zur gesammelten Auflistung aller ICs mit derselben Funktion, usw.). Die BAE-Stückliste wird auf eine datei mit der Endung `.pl` ausgegeben. Die CSV-/DBF-ASCII-Ausgabe erfolgt auf eine Datei mit der Endung `.csv` und listet jedes Bauteil in einer Zeile unter Angabe der Bauteiltypkennung, des Bauteilnamens, der Gehäusebauform und der Attributeeinträge auf. Hierbei wird ein Strichpunkt zur Trennung von Bauteildaten verwendet.

PSTKDRL (LAY) -- Padstack-/Bohrdefinitionsreport

Das Programm **pstkdrl.ulc** erzeugt für eine selektierbare DDB-Datei aus dem aktuellen Verzeichnis eine Liste der Padstack- bzw. Bohrungs-Definitionen und zeigt einen entsprechenden Report in einem Popupmenü mit Ausgabeoption an.

ROUTINFO (LAY) -- Routingdatenanalyse

Das Programm **routinfo.ulc** führt ein ausführlichen Leiterlängenanalyse für das aktuelle Layout durch und speichert nach Bedarf einen entsprechenden Report mit dem Layout. Dieser Report kann später zur Beurteilung der Resultate von Routerläufen herangezogen werden. Hiermit ist insbesondere auch die Prüfung kritischer Netze z.B. nach dem Re-Entrant-Routing im Zuge eines Redesign möglich.

ROUTING (LAY) -- Routingdatenausgabe

Das Programm **routing.ulc** gibt die Leiterbahndaten des aktuell geladene Layout aus.

TBDVLAY (LAY) -- Layout-Entwurfsansichten in Toolbar verwalten

Das Programm **tbdvlay.ulc** bearbeitet Anforderungen zur Aktualisierung von Layout-Entwursfansichten in der mit **TOOLBAR** aktivierten Werkzeugleiste des Layoutsystems.

TESTDATA (LAY) -- Testdatenausgabe

Das Programm **testdata.ulc** erzeugt für das aktuell geladene Layout eine Testdaten-Ausgabe in einem generischen Format.

TRACEREP (LAY) -- Trace Report

Das Programm **tracerep.ulc** zeigt statistische Daten über die Leiterbahnen des aktuell geladenen Layouts (Leiterbahnlängen, Viaanzahl, usw.) in Form eines Reports in einem Popupmenü mit Ausgabeoption an.

UNCONPIN (LAY) -- Anzeige nicht angeschlossener Pins

Das Programm **unconpin.ulc** listet die offenen, d.h. noch nicht angeschlossenen Pins des aktuellen Layouts in einem Popupmenü mit Ausgabeoption auf.

WRLOUT (LAY) -- Layout WRL/VRML-3D-Datenausgabe

Das Programm **wrlout.ulc** erzeugt aus dem aktuell geladenen Layout eine 3D-Datenausgabe im Format WRL (VRML).

4.2.4 GED-Programme

Die nachfolgend aufgeführten **User Language**-Programme sind in der Interpreterumgebung des **Layouteditors** ablauffähig.

AUTONAME (GED) -- Automatische Bauteilbenennung/-Nummerierung

Das Programm **autoname.ulc** ermöglicht die Aktivierung verschiedener Funktionen zur automatischen Umbenennung von Bauteilen auf dem aktuell geladenen Layout. Die Nummerierung der Bauteile erfolgt von links oben nach rechts unten. Um eine lückenlose Nummerierung auch bei der Auswahl identischer Quell- und Zielnamensprefixe zu gewährleisten, wird ein Multi-Pass-Algorithmus für die Umbenennung angewendet.

CONCONV (GED) -- Physikalische Netzliste einlesen

Das Programm **conconv.ulc** dient dazu, physikalische (d.h. gepackte) Netzlisten aus unterschiedlichen ASCII-Formaten (BAE, ALGOREX, Applicon, CADNETIX, CALAY, EEDESIGNER, Marconi ED, Mentor, MULTIWIRE, OrCAD, PCAD, RINF, SCICARDS, TANGO, VECTRON, VUTRAX, WIRELIST) in den **Bartels AutoEngineer** zu übertragen. Nach Auswahl der Netzlistendatei `<project>.con` und der für die Umsetzung zu verwendenden Layoutbibliothek werden die Netzlistendaten eingelesen und in Form einer (pseudo-physikalischen) logischen Netzliste in der DDB-Datei `<project>.ddb` unter dem in der Eingabedatei spezifizierten Plannamen (bzw. unter dem über **BSETUP** mit LAYDEFELEMENT eingestellten Default-Layoutelementnamen) abgelegt. Diese pseudo-physikalische Netzliste lässt sich dann mit dem **Packager** in eine echte physikalische Netzliste umwandeln, für die anschließend das zugehörige Layout erstellt werden kann. Für Konsistenzprüfungen werden die Pinlisten der in der Netzliste referenzierten Bauteilsymbole benötigt; **CONCONV** sucht in der Zielprojektdatei (`<project>.ddb`) bzw. in der selektierten Layoutbibliothek nach den erforderlichen Gehäusedefinitionen (in der Zielprojektdatei fehlende Bauteilsymbole werden ggf. automatisch aus der Layoutbibliothek in die Zielprojektdatei kopiert). Enthält die einzulesende Netzlistendatei Platzierungsdaten (siehe Netzlistenformate BAE, Mentor, RINF, VUTRAX), und existiert das Ziellayout noch nicht, dann wird die entsprechende Bauteilplatzierung nach vorheriger Generierung des Ziellayouts automatisch übernommen. Bei bereits existierendem Ziellayout werden Platzierungsdaten nur nach vorheriger Rückfrage übernommen; in diesem Fall werden bereits platzierte Bauteile entsprechend den Netzlistenvorgaben umplatziert. Die im Anschluss an die Netzlistenumsetzung benötigten logischen Bibliothekseinträge für den **Packager**-Lauf werden synthetisch generiert (Bauteil-/Gehäusezuweisungen mit 1:1-Pinmapping) und in der Zielprojektdatei abgespeichert. Der Fortgang der Bearbeitung sowie eventuell auftretende Fehler und Warnungen werden am Bildschirm und in einer Protokolldatei mit Namen `bae.log` aufgelistet. Nach erfolgreicher Übernahme der Netzliste erfolgt der Hinweis, dass ein **Packager**-Lauf durchzuführen ist.

CONUTIL (GED) -- Netzlistenfunktionen

Das Programm **conutil.ulc** ermöglicht die Aktivierung einer Reihe von Netzlistenfunktionen wie z.B. Netzdatenreport, Netzhighlight (wahlweise mit Zoom), Anzeige offener Pins, Netzlistenprüfung, usw.

DRCBLOCK (GED/HighEnd) -- Utilities fuer den erweiterten DRC

Das Programm **drcblock.ulc** bietet Funktionen zur Definition und Verwaltung von lagenspezifischen Entwurfsregelblöcken an. Bitte beachten Sie, dass die Anwendung derartiger DRC-Blöcke auf **BAE HighEnd** beschränkt ist.

EDIFOUT (GED) -- EDIF 2.0 Netzlistenausgabe

Das Programm **edifout.ulc** generiert eine Netzliste im Format EDIF 2.0 zur Übergabe an externe PLD-Layout- bzw. Fitterprogramme. **EDIFOUT** liest gepackte (logische) Netzlisten und erzeugt jeweils automatisch ein Layout-Arbeitsblatt zur Extraktion der erforderlichen Daten. Die EDIF-Netzliste enthält eine Bibliotheksbeschreibung, eine Schnittstellenbeschreibung sowie die eigentlichen Netzlistendaten. Die Ausgabe erfolgt auf eine Datei mit der Dateinamensendung `.edf`.

FONTEDIT (GED) -- Fonteditor

Das Programm **fontedit.ulc** ermöglicht die Aktivierung von Routinen zur Bearbeitung von Fontdaten, d.h. zum Editieren von Zeichensätzen. Es stehen Funktionen zum Laden und Schreiben von Fontdaten zur Auswahl. Beim Laden eines Fonts werden die Zeichen eines wählbaren Zeichensatzes aus der Datei `ged.fnt` im BAE-Programmverzeichnis umgewandelt in Polygone und auf die Seite 1 der Dokumentarlage 2 geladen. Diese Polygone können mit den **Layouteditor**-Funktionen zur Flächenbearbeitung manipuliert werden. Mit der Funktion zum Schreiben des Fonts werden die auf der Seite 1 der Dokumentarlage 2 enthaltenen Polygone im BAE-Format für Zeichensatzdaten auf eine ASCII-Datei geschrieben; die so erzeugte Fontdatendatei kann mit dem BAE-Utilityprogramm **FONTCONV** anschließend wieder in die Datei `ged.fnt` transferiert werden.

GED_MS (GED) -- GED Mausaktion

Das Programm **ged_ms.ulc** wird automatisch beim Drücken der linken Maustaste gestartet wenn gerade keine andere **Layouteditor**-Funktion aktiv ist. **GED_MS** aktiviert ein kontextsensitives Funktionsmenü für das Objekt an der aktuellen Mausposition. Ist kein Element geladen, dann werden die Funktionen Element Laden, Neues Element und Mausmodus angeboten.

GED_MSG (GED/HighEnd) -- GED Messagesystem-Aktion

Das Programm **ged_msg.ulc** empfängt und bearbeitet Messages aus anderen **BAE HighEnd** Modulen. Die auszulösenden Aktionen (Bauteilplatzierung, Gruppenselektion, Netzhighlight, Bauteilmengenselektion nach SCM-Symbolgruppe, etc.) sowie die zu bearbeitenden Objekte werden jeweils durch die eingehende Meldung bestimmt.

GED_PLC (GED) -- GED Bauteilplatzierungsaktion

The **ged_plc.ulc** **User Language** program is automatically activated after a part is placed to update the netlist assistent or other part relevant permanent dialog boxes.

GEDBOUND (GED) -- Layout-Arbeitsbereich/Elementgrenzen/Nullpunkt setzen

Das Programm **gedbound.ulc** aktiviert ein Menü mit Funktionen zur Vergrößerung und Verkleinerung der Elementgrenzen des aktuell geladenen Elements und zur Justierung des Layoutnullpunkts auf den Nullpunkt des Systemeingaberasters.

GEDDISP (GED) -- GED-Bilddarstellungsfunktionen

Das Programm **geddisp.ulc** ermöglicht die Aktivierung einer Reihe zusätzlicher Bilddarstellungsfunktionen im **Layouteditor**.

GEDDRAW (GED) -- GED-2D-Zeichnungsfunktionen

Das Programm **geddraw.ulc** ermöglicht die Aktivierung verschiedener 2D-Zeichnungsfunktionen zur Generierung von Kreisen, Rechtecken und Pfeilen, zur Distanzbemaßung sowie zur Linealerzeugung.

GEDGROUP (GED) -- GED-Gruppenfunktionen

Das Programm **gedgroup.ulc** ermöglicht die automatische Selektion bzw. De-Selektion speziell definierter Gruppen von Elementen des aktuell geladenen Layoutelements (alle Objekte eines selektierbaren Typs oder mit spezifischen Attributen, alle fixierten/unfixierten, alle gespiegelten/ungespiegelten, alle auf selektierbarer Lage, alle sichtbaren/unsichtbaren, usw.). Darüber hinaus können Funktionen zur Durchführung von Lagenänderungen für gruppenselektierte Elemente, zum Laden von Gruppen aus anderen Datenbankhierarchieebenen, zum automatischen Kopieren von Gruppen, zum Ändern der Textgröße bzw. der Leiterbreite von gruppenselektierten Texten bzw. Leiterbahnen oder zum Zurücksetzen der Verschiebung von Bauteilnamen und Bauteilattributtexten aktiviert werden.

GEDIO (GED) -- GED-Ein-/Ausgabefunktionen

Das Programm **gedio.ulc** aktiviert im **Layouteditor** ein Menü mit verschiedenen layout-spezifischen Ein-/Ausgabefunktionen. Über das Kommando `addioitem` können in der Datei `bae.ini` im BAE-Programmverzeichnis in einfacher Weise zusätzliche (benutzerspezifische) Import-/Exportfunktionen in das Menü aufgenommen werden.

GEDMACL (GED) -- Layout Makro Laden

Das Programm **gedmacl.ulc** lädt das Bibliothekslement eines mausselektierbaren Bauteils, Padstacks oder Pads des aktuell geladenen Layoutelements in den **Layouteditor**.

GEDPART (GED) -- GED-Bauteil- und Platzierungsfunktionen

Das Programm **gedpart.ulc** ermöglicht die Aktivierung einer Reihe von Bauteil- und Platzierungsfunktionen wie z.B. Bauteilsuche, Platzieren nach Bauteilnamensmuster, Platzieren Bauteilmenge, Bauteilplatzierung mit automatischer Bauteilauswahl und Positionierungsvorschlag, automatische Platzierung selektierbarer Hierarchiegruppen entsprechend zuvor durchgefuehrter Bauteilgruppenplatzierung, Bauteiltausch, Drehen bzw. Rotieren selektierbarer Bauteile, Abfrage von Platzierungsdaten, Einlesen und Ausgeben von Platzierungsdaten, Löschen konstruktiver Bauteile, Erstellung von Platzierungs-Histogrammen, (Bauteil-)Höhen-DRC, Drehwinkelvorgaben für **LROTATE** und **RROTATE**, usw.

GEDPICK (GED) -- Layout Polgygonpickfunktionen (Schnittpunkt/Mittelpunkt)

Das Programm **gedpick.ulc** bietet Funktionen zur Selektion von Polygonschnittpunkten bzw. zur Selektion der Mittelpunkte interpolierter Kreisbögen. Die Funktionsfähigkeit dieses Programms ist nur gewährleistet, wenn es implizit über Tastendruck (z.B. x bzw. X) aufrufbar ist.

GEDPOLY (GED) -- GED-Polygonfunktionen

Das Programm **gedpoly.ulc** ermöglicht die Aktivierung einer Reihe von Polygonbearbeitungsfunktionen wie z.B. die Änderung von Polygonlagen, die Definition bzw. Änderung von Polygontypen, das Auftrennen bzw. Zusammenfassen von Dokumentarlinien, die Definition von Polygonlinienbreiten, die Umwandlung von Polygonecken in Kreisbögen oder Diagonalen, das Kopieren gruppenselektierter Polygone mit Skalierung, die Aktivierung verschiedener 2D-Zeichnungsfunktionen, Vorgaben für Höhen-DRC, usw.

GEDRULE (GED) -- Layout Regelzuweisungsutility

Mit dem Programm **gedrule.ulc** können im **Layouteditor** Regeln an Layoutelemente bzw. Layoutgruppen zugewiesen werden. Um Regelsystemfehler zu vermeiden, sind die zugewiesenen Regeln mit dem Regelsystem-Compiler **RULECOMP** zu definieren bzw. zu kompilieren.

GEDSETUP (GED) -- Layout Editor Setup

Das Programm **gedsetup.ulc** stellt eine Reihe von **Layouteditor**-Parametern und Bilddarstellungsmodi auf Defaultwerte ein.

GEDTEXT (GED) -- GED-Text-/Bohrungsfunktionen

Das Programm **gedtext.ulc** ermöglicht die Aktivierung einer Reihe von Textfunktionen wie z.B. Setzen von Textgrößen, Umwandlung in Klein- bzw. Großschreibung, Konvertierung von Texten in Leiterbahnen, Flächen oder Linien, Setzen von Textlinienbreiten für die Plotausgabe, Erzeugung kreisbogenförmig geschriebener Texte, usw.

GEDTRACE (GED) -- GED-Leiterbahn- und Routingfunktionen

Das Programm **gedtrace.ulc** ermöglicht die Aktivierung einer Reihe von Leiterbahn-, Via- und Netzlistenfunktionen, wie z.B. automatisches Abrunden von Leiterbahnecken, automatisches Kürzen von Leiterbahnenden, Generierung (äquidistanter) paralleler Leiterbahnzüge auf Alternativlagen, Umwandlung von Leiterbahnen in Potentialflächen auf Versorgungslagen, Teardrop-Erzeugung, Änderung von Leiterbreiten, Auftrennen von Leiterbahnsegmenten, Justieren von Leiterbahnsegmentlängen, Netzdatenabfrage, Netz-Highlight (wahlweise mit Zoom), Netzlistenausgabe, Bahnlängenabfrage, Löschen bzw. Neu-Definition von Versorgungslagen, Leiterbahn- bzw. Unroutes-Report, Pin-/Via-Statistik, Vias platzieren, Vias bewegen, Vias löschen, Viatypen ändern, Auswahl des Anzeigemodus für die Leiterbahnbearbeitung, usw.

GEDVAR (GED) -- Layoutvariante selektieren

Das Programm **gedvar.ulc** aktiviert eine selektierbare Layoutvariante.

GEDVIA (GED) -- GED Viafunktionen

Das Programm **gedvia.ulc** ermöglicht die Aktivierung einer Reihe von Viafunktionen, wie z.B. Pin-/Via-Statistik, Vias platzieren, Vias bewegen, Vias löschen, repetitive Viaselektion, mausselektierbare oder gruppenselektierte Viatypen austauschen, Umwandlung von Vias in Bauteile, usw.

GENLMAC (GED) -- Layout-Bibliothekselementerzeugung

Das Programm **genlmac.ulc** ist ein Utility zur halbautomatischen Generierung von Layout-Bibliothekselementen, d.h. von Pad- und Padstackdefinitionen. Mit dem Padgenerator können runde, quadratische, rechteckige, fingerförmige, patronenförmige, oktagonale oder ringförmige Pads sowie Bohrsymbol-Pads definiert werden. Die Benennung der Pads wird automatisch aus der Padform und der Padgröße abgeleitet. Mit dem Padstackgenerator können Standard-Pins (bedrahtet), SMD-Pins, Bohrlöcher (wahlweise durchkontaktiert), Standard-Vias oder partielle Durchkontaktierungen generiert werden. Die Benennung der Padstacks wird automatisch aus dem Padstacktyp und der Pingröße abgeleitet. Die Pingröße ergibt sich dabei aus der Selektion der gewünschten Padform bzw. aus der spezifizierten Bohrlochgröße. Partielle Durchkontaktierungen können für die Signallagen 1 bis 8 definiert werden. Mit dem Bauteilgenerator können Symbole für Widerstände, Kondensatoren und Elektrolytkondensatoren (ELKOs) mit unterschiedlichen Gehäusebauformen (rechteckig/Blockform, zylindrisch/Röhrenform, Platten- bzw. Scheiben- oder Tropfenform) und Anschlussausführungen (radial, axial, axial/stehend) definiert werden. Die Bauteilbenennung wird automatisch aus dem Bauteiltyp, dem Pinabstand, der Bauform und den Gehäuseabmessungen sowie dem Bohr- bzw. Drahtdurchmesser der Anschlüsse abgeleitet.

LAYDXFDI (GED) -- Layout AutoCAD/DXF-Übernahme

Das Programm **laydxfdi.ulc** ermöglicht das Einlesen von Zeichnungsdaten im Format AUTOCAD-DXF auf das aktuell geladene Layoutelement, wobei Eingabe-Längeneinheiten, Einleseoffset und Lagenzuordnung frei wählbar sind.

LAYEDBAT (GED) -- Layoutbibliothek Batcheditor

Das Programm **layedbat.ulc** ermöglicht die Festlegung von Kommandosequenzen zum Editieren bzw. Ändern von Layouts bzw. Layoutsymbolen selektierbarer Dateien des aktuellen Verzeichnisses im Batchbetrieb. **layedbat.ulc** ermöglicht die Aktivierung von Funktionen zum Ändern der Elementgrenzen, zum Tauschen von Padstack- bzw. Padmakros, zur Änderung von lagenzuweisungen, zur Regelzuweisung, zum Definieren, Ändern oder Löschen spezieller Texte, zum Setzen von Bauteilnullpunkten und Bauteilpickpunkten, zum Bewegen bzw. Platzieren von Pinnamen, usw.

LCIFIN (GED) -- Layout CIF-Datenübernahme

Mit dem Programm **lcifin.ulc** können Caltech-CIF-Daten aus einer selektierbaren Datei in das aktuell geladene Layoutelement übernommen werden. Die in der CIF-Datei definierten Strukturen werden auf die aktuelle Layouthierarchieebene geladen und anschließend zur Gruppe selektiert.

LERRLIST (GED) -- Layout DRC-Fehlerliste

Das Programm **lerrlist.ulc** zeigt eine Liste der Designregelverletzungen mit einer Option zum Zoomen auf selektierbare Fehlermarker an.

LLIBDOC (GED) -- Layout-Bibliotheksdokumentation

Das Programm **llibdoc.ulc** erzeugt in selektierbaren Layout-Bibliotheksdateien des aktuellen Verzeichnisses Layoutelemente, auf die sortiert nach Namen die in der jeweiligen DDB-Datei enthaltenen Layoutsymbole automatisch platziert werden. Die so erzeugten Layoutpläne können (z.B. mit dem **User Language**-Programm **LAYEPS**) ausgeplottet und als Bibliotheks-Dokumentation (Bauteilmappe) verwendet werden. Die Dokumentation kann wahlweise für Bauteil-, Padstack- oder Padsymbole erstellt werden, wobei eine Selektion über Symbolnamensmuster möglich ist. Es stehen unterschiedliche Blattformate für die Ausgabe zur Verfügung.

LMACREAD (GED) -- Layout Makrodefinition Import

Das Programm **lmacread.ulc** importiert Layoutmakrodefinitionen von einer Textdatei in den **Layouteditor**.

LPINTRC (GED) -- Layout Pin Leiterbahnanbindung

Das Programm **lpintrc.ulc** ermöglicht das automatische Anrouten an rasterlos platzierte Pins, Vias oder Leiterbahnen während des manuellen Routings. Diese Funktion steht nur während des manuellen Verlegens von Leiterbahnen bei der Bearbeitung des Start- oder des Endpunktes der Leiterbahn zur Verfügung und muss über einen Hotkey (z.B. **p** oder **P**) aktiviert werden. Bei Bearbeitung des Leiterbahnstartpunktes wird eine Leiterbahnecke auf den Ursprung des Pins gelegt, auf dem sich der Grafikkursor gerade befindet. Bei Bearbeitung des Leiterbahnendpunktes wird ausgehend vom letzten Leiterbahnpunkt ein 45-Grad-Segment eingefügt und anschließend eine gerade Bahn zum Pinursprung gezogen. In beiden Fällen bleiben die aktuellen Raster- und Winkeleinstellungen erhalten.

LSYMEDIT (GED) -- Layoutbauteil Edititierfunktionen

Das Programm **lsymedit.ulc** ermöglicht bei geladenem Layoutsymbol den menügesteuerten Aufruf einer Reihe von Symbol-Editierfunktionen. Zur Auswahl stehen unter anderem Routinen zum Setzen des Symbol-Nullpunkts oder des Bestückpicks, zum Platzieren von Pinlisten bzw. Pinreihen, zur Änderung der Elementgrenzen, usw.

LTXFIN (GED) -- TXF Layout-Datenübernahme

Das Programm **ltxfin.ulc** importiert TXF-Layoutdaten aus einer selektierbaren ASCII-Datei in den **Layouteditor**

MT_ROUT (GED) -- Mikami-Tabuchi Router

Das Programm **mt_rout.ulc** führt für zwei mausselektierbare Verbindungspunkte ein Linien-Autorouting nach dem Mikami-Tabuchi-Algorithmus durch. Während der Verbindungspunktselektion kann über die rechte Maustaste ein Kontextmenü für Routereinstellungen aktiviert werden. Die Aufrufsequenz `mt_rout:'param'` aktiviert anstatt eines Routinglaufs einen Dialog zur Einstellung der Routingparameter.

POLYRND (GED) -- Polygonecken abschrägen/runden

Mit dem Programm **polyrnd.ulc** können Polygonecken automatisch in Kreisbögen oder 45-Grad-Segmente umgewandelt werden.

READLPLC (GED) -- Layout-Platzierungsdatenübernahme

Das Programm **readlplc.ulc** liest aus einer selektierbaren ASCII-Datei Platzierungsdaten in einem der durch das **User Language**-Programm **DUMPPLC** erzeugten Formate ein und führt automatisch die darin definierte Bauteilplatzierung durch.

TEARDROP (GED) -- Teardropfunktionen

Das Programm **teardrop.ulc** erzeugt automatisch für alle auf Durchkontaktierungen bzw. Bauteilpins liegenden Leiterbahnenden sogenannte Teardrops, d.h. fließende Verbreiterungen der Leiterbahnenden auf den Durchmesser des Vias bzw. des Pins. Die Teardropgenerierung kann wahlweise auf (selektierbare) Bauteilanschlüsse oder Durchkontaktierungen beschränkt werden. Neben der Generierung von Teardrops besteht auch die Möglichkeit, Teardrops zu selektieren bzw. zu löschen.

TRACERND (GED) -- Leiterbahnecken abschraegen/runden

Das Programm **tracernd.ulc** rundet die Leiterbahnecken selektierbarer Leiterbahnen des aktuell geladenen Layouts automatisch ab. Der Radius für die Abrundungen ist dabei frei wählbar.

TRCPUSH (GED) -- Leiterbahnbuendel verschieben

Das Programm **trcpush.ulc** verschiebt maus-selektierbare Leiterbahnsegmente um einen frei wählbaren Verschiebevektor. Benachbarte Bahnen bzw. Bahnsegmente auf der selben Lage werden nach Bedarf ebenfalls verschoben, um genügend Platz zur Verschiebung des selektierten Bahnsegments zu schaffen. Um sicherzustellen, dass das Leiterbild der bearbeiteten Bahnen (d.h. die Orientierung der Leiterbahnsegmente) unverändert bleibt, werden adjazente Segmente nach Bedarf mitverschoben bzw. verlängert oder verkürzt. Damit eignet sich **trcpush.ulc** zur Verschiebung ganzer Leiterbahnbündel (Push'n'Shove).

VHDLOUT (GED) -- VHDL Netzlistenausgabe

Das Programm **vhdlout.ulc** generiert eine Netzliste im Format VHDL zur Übergabe an externe PLD-Layout- bzw. Fitterprogramme. **EDIFOUT** liest gepackte (logische) Netzlisten und erzeugt jeweils automatisch ein Layout-Arbeitsblatt zur Extraktion der erforderlichen Daten. Für hierarchische Designs werden Modulports in der VHDL-Ausgabe erzeugt. Die Ausgabe erfolgt auf eine Datei mit der Dateinamensendung **`.vhdl`**.

4.2.5 Autorouter-Programme

Die nachfolgend aufgeführten **User Language**-Programme sind in der Interpreterumgebung des **Autorouters** ablauffähig.

AR_MS (AR) -- Autorouter Mausaktion

Das Programm **ar_ms.ulc** wird automatisch beim Drücken der linken Maustaste gestartet wenn gerade keine andere **Autorouter**-Funktion aktiv ist.. **AR_MS** aktiviert ein kontextsensitives Funktionsmenü für das Objekt an der aktuellen Mausposition. Ist kein Element geladen, dann wird die Funktion Element Laden angeboten.

ARDISP (AR) -- Autorouter-Bilddarstellungsfunktionen

Das Programm **ardisp.ulc** ermöglicht die Aktivierung einer Reihe zusätzlicher Bilddarstellungsfunktionen im **Autorouter**.

ARIO (AR) -- Autorouter-Ein-/Ausgabefunktionen

Das Programm **ario.ulc** aktiviert im **Autorouter** ein Menü mit verschiedenen layout-spezifischen Ein-/Ausgabefunktionen. Über das Kommando **`addioitem`** können in der Datei **`bae.ini`** im BAE-Programmverzeichnis in einfacher Weise zusätzliche (benutzerspezifische) Import-/Exportfunktionen in das Menü aufgenommen werden.

ARSETUP (AR) -- Autorouter Setup

Das Programm **arsetup.ulc** stellt eine Reihe von **Autorouter**-Parametern und Bilddarstellungsmodi auf Defaultwerte ein.

4.2.6 CAM-Prozessor-Programme

Die nachfolgend aufgeführten **User Language**-Programme sind in der Interpreterumgebung des **CAM-Prozessors** ablauffähig.

CAM_MS (CAM) -- CAM-Prozessor Mausaktion

Das Programm **cam_ms.ulc** wird automatisch beim Drücken der linken Maustaste gestartet wenn gerade keine andere **CAM-Prozessor**-Funktion aktiv ist. **cam_ms.ulc** aktiviert ein kontextsensitives Funktionsmenü für das Objekt an der aktuellen Mausposition. Ist kein Element geladen, dann wird die Funktion Element Laden angeboten.

CAMBATCH (CAM) -- CAM-Datenausgabe im Batch-Betrieb

Das Programm **cambatch.ulc** führt eine CAM-Datenausgabe im Batchbetrieb durch und ist Beispiel bzw. Vorlage zur Anpassung an firmenspezifische Bedürfnisse gedacht. Es werden in einem einzigen Programmlauf Gerberdaten für eine Reihe von Signal-, Versorgungs- und Dokumentarlagen sowie Bohrdaten erzeugt.

CAMBATDB (CAM) -- CAM-Batchdatenbank

Das Programm **cambatdb.ulc** dient der Verwaltung einer Datenbank mit benutzerdefinierten CAM-Batchausgabefunktionen unter Windows und Motif.

CAMIO (CAM) -- CAM-Prozessor-Ein-/Ausgabefunktionen

Das Programm **camio.ulc** aktiviert im **CAM-Prozessor** ein Menü mit verschiedenen layout-spezifischen Ein-/Ausgabefunktionen. Über das Kommando **`addioitem`** können in der Datei **`bae.ini`** im BAE-Programmverzeichnis in einfacher Weise zusätzliche (benutzerspezifische) Import-/Exportfunktionen in das Menü aufgenommen werden.

CAMSETUP (CAM) -- CAM-Prozessor Setup

Das Programm **camsetup.ulc** stellt eine Reihe von **CAM-Prozessor**- und Gerberplot-Parametern und Bilddarstellungsmodi auf Defaultwerte ein und aktiviert anhand einer Tabelle Stiftzuordnungen für Multilagenplots.

GAPTUTIL (CAM) -- Gerber-Blendentabellenverwaltung

Das Programm **gaptutil.ulc** ermöglicht die Aktivierung von Utilities zur Verwaltung von Gerber-Blendentabellen wie z.B. die Ein- und Ausgabe von Blendentabellen im BAE- bzw. ECAM-ASCII-Format.

GBALLSIG (CAM) -- Gerber-Signallagenausgabe im Batch-Betrieb

Das Programm **gballsig.ulc** erzeugt im Batch-Betrieb Gerberdaten-Ausgaben für alle Signallagen.

GINSOUT (CAM) -- Generische Bestückdatenausgabe

Das Programm **ginsout.ulc** erzeugt eine generische Bestückdatenausgabe für das aktuell geladene Layout. Die Definition des Ausgabeformats wird aus einer externen Formatanweisungsdatei mit der Dateinamensendung .ifs (Insertion Format Specification) geladen. Die Bestückdatenkoordinaten werden in Bezug zum aktuell eingestellten CAM-Nullpunkt ausgegeben.

POWDCHK (CAM) -- Versorgungslagen Waermefallenpruefung

Das Programm **powdchk.ulc** ermittelt die Koordinaten all jener Wärmefallen, die möglicherweise durch in der Nähe liegende Bohrlochaussparungen vom Rest der Versorgungslage isoliert sind. Die Ausführung der hierbei angezeigten Wärmefallen sollte vor einer Fertigungsdatenausgabe in jedem Fall nochmals geprüft und ggf. korrigiert werden.

4.2.7 CAM-View-Programme

Die nachfolgend aufgeführten **User Language**-Programme sind in der Interpreterumgebung des **CAM-View**-Moduls ablauffähig.

CV_MS (CV) -- CAM-View Mausaktion

Das Programm **cv_ms.ulc** wird automatisch beim Drücken der linken Maustaste gestartet wenn gerade keine andere **CAM-View**-Funktion aktiv ist. **cv_ms.ulc** aktiviert ein kontextsensitives Funktionsmenü für das Objekt an der aktuellen Mausposition. Sind keine CAM-Daten geladen, dann werden ausführbare Funktionen aus dem Menü Datei angeboten.

CVSETUP (CV) -- CAM View Setup

Das Programm **cvbatld.ulc** ermöglicht das automatisierte Einlesen und Ausschreiben von mit dem Programm **CAMBATDB** im Batchbetrieb erstellten Ausgabedateien.

CVSETUP (CV) -- CAM View Setup

Das Programm **cvsetup.ulc** stellt eine Reihe von **CAM-View**-Parametern und Bilddarstellungsmodi auf Defaultwerte ein.

4.2.8 IC-Design-Programme

Die nachfolgend aufgeführten **User Language**-Programme sind in der Interpreterumgebung des **Chipeditors** ablauffähig.

CHKIMAC (ICD) -- Nicht definierte IC-Design-Makroreferenzen auflisten

Das Programm **chkimac.ulc** listet die Namen aller undefinierten Makroreferenzen des aktuell geladenen **IC-Design**-Elements in einem Popupmenü mit Ausgabeoption auf.

CHECKINL (ICD) -- IC-Design gegen Netzliste pruefen

Das Programm **checkinl.ulc** prüft das aktuelle IC-Layout gegen die Netzlistendaten und zeigt einen Report über noch nicht platzierte Zellen, falsche Zellentypen und fehlende Pindefinitionen in einem Popupmenü mit Ausgabeoption an.

ICDPCR (ICD) -- IC-Design-Report

Das Programm **icdpcr.ulc** zeigt einen Report über das aktuell geladene **IC-Design**-Element in einem Popupmenü mit Ausgabeoption an.

4.2.9 CED-Programme

Die nachfolgend aufgeführten **User Language**-Programme sind in der Interpreterumgebung des **Chipeditors** ablauffähig.

CED_MS (CED) -- Chipeditor Mausaktion

Das Programm **ced_ms.ulc** wird automatisch beim Drücken der linken Maustaste gestartet wenn gerade keine andere **Chipeditor**-Funktion aktiv ist. **ced_ms.ulc** aktiviert ein kontextsensitives Funktionsmenü für das Objekt an der aktuellen Mausposition. Ist kein Element geladen, dann werden die Funktionen Element Laden, Neues Element und Mausmodus angeboten.

CEDDISP (CED) -- Chipeditor-Bilddarstellungsfunktionen

Das Programm **ceddisp.ulc** ermöglicht die Aktivierung einer Reihe zusätzlicher Bilddarstellungsfunktionen im **Chipeditor**.

CEDGROUP (CED) -- Chipeditor-Gruppenfunktionen

Das Programm **cedgroup.ulc** ermöglicht die automatische Selektion bzw. De-Selektion speziell definierter Gruppen von Elementen des aktuell geladenen **IC-Design**-Elements (alle Objekte eines selektierbaren Typs, alle fixierten/unfixierten, alle gespiegelten/ungespiegelten, alle auf selektierbarer Lage, usw.). Darüber hinaus können Funktionen zum automatischen Kopieren von Gruppen, zum Löschen lagenselektierter Gruppen oder zum Ändern der Textgröße bzw. der Leiterbreite von gruppenselektierten Texten bzw. Leiterbahnen aktiviert werden.

CEDMACL (CED) -- IC Design Makro Laden

Das Programm **cedmacl.ulc** l&aeuml;dt das Bibliothekslement einer mausselektierbaren IC-Zelle oder eines IC-Pins des aktuell geladenen **IC-Design**-Elements in den **Chipeditor**.

CEDPART (CED) -- Chipeditor-Makro- und Platzierungsfunktionen

Das Programm **cedpart.ulc** ermöglicht die Aktivierung einer Reihe von Makro- und Platzierungsfunktionen wie z.B. Zellenplatzierung über Makronamensmuster, Makrotausch, Drehen bzw. Spiegeln selektierbarer Makros, Einlesen und Ausgeben von Platzierungsdaten, Löschen konstruktiver Makros, Abfrage von Platzierungsdaten, usw.

CEDPOLY (CED) -- Chipeditor-Polygonfunktionen

Das Programm **cedpoly.ulc** ermöglicht die Aktivierung einer Reihe von Polygonbearbeitungsfunktionen wie z.B. die Änderung von Polygonlagen, die Definition bzw. Änderung von Polygontypen, das Kopieren gruppenslektierter Polygone mit Skalierung, die Aktivierung verschiedener 2D-Zeichnungsfunktionen, usw.

CEDSETUP (CED) -- Chipeditor Setup

Das Programm **cedsetup.ulc** stellt eine Reihe von **Chipeditor**-Parametern und Bilddarstellungsmodi auf Defaultwerte ein.

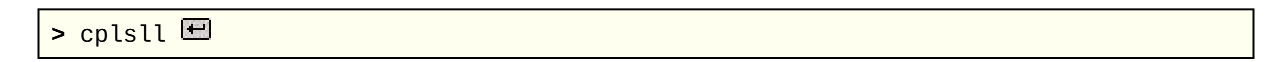
4.3 Bereitstellung der User Language-Programme

Dieser Abschnitt gibt Hinweise zur Installation der **User Language**-Programme der BAE-Software und beschreibt die Möglichkeiten der Menübelegung und Tastaturprogrammierung.

Mit der BAE-Software werden eine Vielzahl von **User Language**-Programmen in kompilierter Form in der Datei **`ulcprog.vdb`** im BAE-Programmverzeichnis installiert. Zusätzlich werden sämtliche **User Language**-Programme im *Quellcode* in einem speziell hierfür vorgesehenen Verzeichnis (**`baeulc`**) bereitgestellt. Eine komplette Auflistung mit Kurzbeschreibungen aller **User Language**-Programme finden Sie in Kapitel 4.2.

4.3.1 Kompilierung

Die Kompilierung der mit der BAE-Software ausgelieferten **User Language**-Programme ist üblicherweise nicht notwendig, da die Programme bereits in kompilierter Form installiert werden. Nichtsdestotrotz wird mit den **User Language**-Programmen unter anderem die Batchdatei CPLSLL (ComPiLe with Static Link Library) zur automatischen Kompilierung sämtlicher **User Language**-Programme aus dem **User Language**-Verzeichnis bereitgestellt. Unter DOS kann CPLSLL nach dem Setzen der **`PATH`**-Variable im **User Language**-Verzeichnis (**`baeulc`**) mit



```
> cplsll ↵
```

gestartet werden. Der entsprechende Aufruf unter Linux bzw. Unix lautet

```
> cplsll.bat ↵
```

Der Übersetzungsvorgang kann (je nach Leistungsfähigkeit des Rechners) einige Zeit in Anspruch nehmen.

4.3.2 Menübelegung und Tastaturprogrammierung

Einige der installierten **User Language**-Programme definieren implizite **User Language**-Programmaufrufe über die eine weit reichend modifizierte Benutzeroberfläche mit einer Vielzahl von Zusatzfunktionen (Startups, Toolbars, Menübelegung, Tastaturprogrammierung) aktiviert wird. Selbstverständlich haben Sie die Möglichkeit, weitere Anpassungen selbst vorzunehmen oder die vorgegebene Menü- und Tastaturbelegegung ganz oder teilweise zurückzusetzen.

Das mit der BAE-Software ausgelieferte **User Language**-Startupprogramm **BAE_ST** wird automatisch beim Aufruf eines BAE-Moduls mit integriertem **User Language Interpreter** (**Schaltplaneditor**, **Layouteditor**, **Autorouter**, **CAM-Prozessor**, **CAM-View**, **Chipeditor**) gestartet. **BAE_ST** ruft seinerseits das **User Language**-Programm **UIFSETUP** auf, welches eine vordefinierte Menü- und Tastaturbelegung im aktuellen BAE-Programm-Modul aktiviert. Änderungen bzw. Anpassungen der BAE-Menü- und Tastaturbelegung können *zentral* in der Quellcodedatei des Programms **UIFSETUP** vorgenommen werden. Die aktuelle Tastaturbelegung kann mit dem **User Language**-Programm **HLPKEYS** angezeigt werden. Der Aufruf von **HLPKEYS** ist über die Funktion Tastaturbelegung aus dem Menü Hilfe möglich, sofern die vordefinierte Menübelegung aus **UIFSETUP** aktiviert ist. Die Transparenz der mit **UIFSETUP** definierten Menübelegung ist automatisch durch die Anzeige der entsprechenden BAE-Menüs gegeben. Darüber hinaus kann mit dem Programm **UIFDUMP** die in der aktuellen Interpreterumgebung definierte Menü- und Tastaturbelegung in Form eines Reports angezeigt bzw. auf eine ASCII-Datei ausgegeben werden. Mit dem Programm **UIFRESET** lässt sich die komplette Menü- und Tastaturbelegung der aktuellen Interpreterumgebung zurücksetzen. Das Ergebnis eines **UIFRESET**-Aufrufs wird Sie sicher überraschen (probieren Sie es doch einfach einmal aus; durch einen Aufruf von **UIFSETUP** können Sie ja jederzeit wieder die modifizierte Benutzeroberfläche aktivieren). Die Programme **UIFSETUP**, **UIFDUMP** und **UIFRESET** sind zusätzlich auch über das Menü des Programms **KEYPROG** aufrufbar, welches zudem komfortable Funktionen zur Online-Tastaturprogrammierung sowie zur Verwaltung von Hilfstexten für **User Language**-Programme zur Verfügung stellt.

Anhang A
Konventionen und Definitionen

Dieser Anhang beschreibt die Konventionen für den Zugriff auf die in der **User Language** definierten Index-Variablen-Typen und Systemfunktionen sowie die hierfür definierten Wertebereiche. Dabei erfolgt zunächst die Definition der Begriffe Interpreterumgebung und Aufruftyp sowie anschließend geordnet nach den jeweiligen Aufruftypen die Auflistung der Wertebereichsdefinitionen.

Inhalt

Tabellen

A.1 Konventionen

Die in diesem Abschnitt festgelegten Konventionen sind für den Zugriff auf die in der **User Language** definierten Index-Variablen-Typen und Systemfunktionen von essentieller Bedeutung.

A.1.1 Interpreterumgebung

Die Interpreterumgebung bezeichnet den Programm-Teil des **Bartels AutoEngineer**, in dem der **User Language Interpreter** eingebunden ist. Je nach Interpreterumgebung können unterschiedliche Index-Variablen-Typen und Systemfunktionen implementiert und damit verfügbar sein. Ein **User Language**-Programm ist jeweils nur in der Interpreterumgebung ablauffähig, in der alle im Programm referenzierten Index-Variablen-Typen und Systemfunktionen implementiert sind.

A.1.2 Aufruftyp

Damit sowohl der **User Language Compiler**, als auch der **User Language Interpreter** möglichst frühzeitig Kompatibilitätsprobleme erkennen können, ist in der **User Language** der Begriff des Aufruftyps definiert. Jedem Index-Variablen-Typ und jeder Systemfunktion der **Bartels User Language** ist ein Aufruftyp zugeordnet, der in kodierter Form die Interpreterumgebung(en) definiert, in der das jeweilige Objekt implementiert ist. Mit Hilfe dieser Aufruftyp-Definitionen ist der **User Language Compiler** in der Lage, ein **User Language**-Programm daraufhin zu prüfen, ob zueinander inkompatible Referenzen in diesem Programm enthalten sind. Das Ergebnis dieser Prüfung ist ein dem erzeugten Maschinenprogramm zugeordneter Aufruftyp, der die Interpreterumgebungen definiert, in denen das Programm ablauffähig ist, und mit dessen Hilfe der **User Language Interpreter** wiederum entscheiden kann, ob ein zu startendes Maschinenprogramm zur aktuellen Interpreterumgebung kompatibel ist.

Tabelle A-1 enthält eine Liste aller definierten Aufruftypen. Jeder Index-Variablen-Typ und jede Systemfunktion der **Bartels User Language** ist genau einem dieser Aufruftypen zugeordnet.

Tabelle A-1: User Language Aufruftypen

Code	Aufruftyp-Bezeichnung
STD	Standard
CAP	Schematic Capture Datenzugriff
SCM	**Schaltplaneditor**
LAY	Layout Datenzugriff
GED	**Layouteditor**
AR	**Autorouter**
CAM	**CAM-Prozessor**
CV	**CAM-View**
ICD	IC Design Datenzugriff
CED	**Chipeditor**

Tabelle A-2 enthält die Information über die Kompatibilität der in der **Bartels User Language** definierten Aufruftypen zueinander. Entsprechend dieser Tabelle führt der **User Language Compiler** die Kompatibilitätsprüfung für die innerhalb eines Programms enthaltenen Systemreferenzen durch. Wird ein Objekt eines bestimmten Aufruftyps in einem Programm verwendet, so müssen auch alle anderen in diesem Programm verwendeten Objekte der Menge der entsprechend als kompatibel gekennzeichneten Aufruftypen zugeordnet sein (sofern nicht die Preprozessoranweisung `#pragma ULCALLERSTD` verwendet wird; siehe hierzu Abschnitt 2.6.5).

Tabelle A-2: Kompatibilität Aufruftyp zu Aufruftyp

Aufruftyp	STD	CAP	SCM	LAY	GED	AR	CAM	CV	ICD	CED
STD	x	x	x	x	x	x	x	x	x	x
CAP	x	x	x	-	-	-	-	-	-	-
SCM	x	x	x	-	-	-	-	-	-	-
LAY	x	-	-	x	x	x	x	-	-	-
GED	x	-	-	x	x	-	-	-	-	-
AR	x	-	-	x	-	x	-	-	-	-
CAM	x	-	-	x	-	-	x	-	-	-
CV	x	-	-	-	-	-	-	x	-	-
ICD	x	-	-	-	-	-	-	-	x	x
CED	x	-	-	-	-	-	-	-	x	x

Tabelle A-3 enthält die Information über die Kompatibilität der in der **Bartels User Language** definierten Aufruftypen zu den Interpreterumgebungen. Entsprechend dieser Tabelle führt der **User Language Interpreter** die Kompatibilitätsprüfung für die in einer Interpreterumgebung zu startenden **User Language**-Programme durch.

Tabelle A-3: Kompatibilität Aufruftyp zu Interpreter

Aufruftyp	Interpreter					
	SCM	GED	AR	CAM	CV	CED
STD	x	x	x	x	x	x
CAP	x	-	-	-	-	-
SCM	x	-	-	-	-	-
LAY	-	x	x	x	-	-
GED	-	x	-	-	-	-
AR	-	-	x	-	-	-
CAM	-	-	-	x	-	-
CV	-	-	-	-	x	-
ICD	-	-	-	-	-	x
CED	-	-	-	-	-	x

A.2 Wertebereichsdefinitionen

In der **User Language** sind für eine Reihe von Index-Variablen-Typ-Elementen und Systemfunktions-Parametern Wertebereiche definiert. Dieser Abschnitt enthält eine komplette Übersicht über diese Definitionen. In der Beschreibung für die Index-Variablen-Typen (siehe Anhang B) und Systemfunktionen (siehe Anhang C) werden die hier definierten Wertebereiche wo immer nötig durch die entsprechende Benennung für den Wertebereich referenziert.

A.2.1 Standard Wertebereiche (STD)

Die nachfolgend aufgeführten Kodierungen besitzen Gültigkeit in den Definitionen für den Aufruftyp STD. Sie definieren mithin gültige Wertebereiche für spezielle Elemente von Index-Variablen-Typen bzw. Systemfunktions-Parameter innerhalb aller Interpreterumgebungen. Die Benennung der Wertebereichsdefinitionen ergibt sich aus dem Schlüsselwort "STD" und einer fortlaufenden Nummer.

STD1 - Datenbankklassencodes:

```
(-1) = Unbekannte/ungültige Datenbankklasse
 100 = Layout Leiterkarte
 101 = Layout Bauteil
 102 = Layout Padstack
 103 = Layout Pad
 150 = Layout Leiterbahnen
 151 = Layout Connectivity
 160 = Layout DRC Parameter
 200 = Physikalische Netzliste
 201 = Physikalische Zuweisungen
 202 = Backannotation Anforderung
 300 = Autorouter Daten
 301 = Autorouter Parameter
 400 = BAE Zeichensatz
 401 = BAE Setup Daten
 402 = Parameter Setup Daten
 500 = Gerber Blendentabelle
 501 = Layout Farbtabelle
 502 = SCM Farbtabelle
 510 = IC Farbtabelle
 511 = GDS Struktur
 700 = BAE Meldungen
 800 = SCM Stromlaufblatt
 801 = SCM Symbol
 802 = SCM Marker
 803 = SCM Label
 850 = SCM Bauteilliste
 900 = Logische Bibliothek
 901 = Logische Netzliste
 902 = Physikalische Pin Info
 903 = Packager Parameter
1000 = IC Layout
1001 = IC Zelle
1002 = IC Pin
1050 = IC Leiterbahnen
1051 = IC Connectivity
1052 = IC Autorouter Daten
1200 = User Language Programm
1201 = User Language Library
1400 = Design-Regel
1401 = Designregel-Liste
1402 = Designregel-Quellcode
4096 = SQL Info Tabellenstruktur
4097 = SQL Info freie Tabellen
4352 = SQL anwenderspezifische Tabelle
   : = :
8191 = SQL anwenderspezifische Tabelle
```

STD2 - Längen-Einheiten:

```
Längenangaben erfolgen - soweit nicht anders vermerkt - in der Einheit Meter.
```

STD3 - Winkel-Einheiten:

```
Winkelangaben erfolgen - soweit nicht anders vermerkt - in der Einheit Bogenmaß.
```

STD4 - Menüpunktnummern:

```
[   0,  99] = Hauptmenüpunkte
[ 100, 199] = Menüpunkte Menü 1
[ 200, 299] = Menüpunkte Menü 2
[ 300, 399] = Menüpunkte Menü 3
[ 400, 499] = Menüpunkte Menü 4
[ 500, 599] = Menüpunkte Menü 5
[ 600, 699] = Menüpunkte Menü 6
[ 700, 799] = Menüpunkte Menü 7
[ 800, 899] = Menüpunkte Menü 8
[ 900, 999] = Menüpunkte Menü 9
[1000,1099] = Menüpunkte Menü 10
       9003 = Menüfunktion Undo
       9004 = Menüfunktion Redo
       9005 = Aktuell geladenes Element schließen
       9006 = Sprung zu Schaltplaneditor
       9007 = Sprung zu Packager
       9008 = Sprung zu Packager mit automatischem Lauf und
                Sprung zu Schaltplaneditor wenn kein Fehler aufgetreten ist
       9009 = Sprung zu Packager mit automatischem Lauf und
                Sprung zu Layouteditor wenn kein Fehler aufgetreten ist
       9010 = Sprung zu Layouteditor
       9012 = Sprung zu Autorouter
       9013 = Sprung zu CAM-Prozessor
       9014 = Sprung zu CAM-View
       9015 = BAE HighEnd/IC Design: Sprung zu IC Design Chip Editor
       9016 = BAE HighEnd/IC Design: Sprung zu IC Design CIF-View
       9017 = BAE HighEnd/IC Design: Sprung zu IC Design GDS-View
       9018 = BAE HighEnd/IC Design: Sprung zu IC Design Cell Placer
       9020 = Sprung zu BAE-Setup
       9021 = Sprung zu BAE-Hauptmenü
       9022 = BAE HighEnd: Aufruf Schaltplaneditor
       9023 = BAE HighEnd: Aufruf Packager
       9024 = BAE HighEnd: Aufruf Packager mit automatischem Lauf
       9025 = BAE HighEnd: Aufruf Layouteditor
       9027 = BAE HighEnd: Aufruf Autorouter
       9028 = BAE HighEnd: Aufruf CAM-Prozessor
       9029 = BAE HighEnd: Aufruf CAM-View
       9030 = BAE HighEnd/IC Design: Aufruf IC Design Chip Editor
       9031 = BAE HighEnd/IC Design: Aufruf IC Design CIF-View
       9032 = BAE HighEnd/IC Design: Aufruf IC Design GDS-View
       9033 = BAE HighEnd/IC Design: Aufruf IC Design Cell Placer
       9035 = BAE HighEnd: Aufruf BAE-Hauptmenü
       9036 = Menüfunktion Programmende
       9038 = Menüfunktion Hilfe
       9039 = Menüfunktion Hilfe zu
       9041 = Sprung zu Packager mit automatischem Lauf und
                Sprung zu Chipeditor wenn kein Fehler aufgetreten ist
       9042 = Löschen nach Zwischenablage
       9043 = Kopieren nach Zwischenablage
       9044 = Einfügen aus Zwischenablage
       9048 = Programm starten und auf dessen beendigung warten
       9049 = Programm starten und sofort zurückkehren
```

Windows-/Motif-Dialoge:

```
        5000 = Schaltplaneditor: Dialog Bilddarstellungsparameter
        5001 = Schaltplaneditor: Dialog allgemeine SCM-Parameter
        5002 = Schaltplaneditor: Dialog SCM-Plotparameter
        5005 = Layouteditor: Dialog Bilddarstellungsparameter
        5006 = Layouteditor: Dialog allgemeine Layouteditor-Parameter
        5008 = Layouteditor: Dialog Flächenfüllparameter
        5010 = Layouteditor: Dialog Platzierungsparameter
        5000 = Autorouter: Dialog Bilddarstellungsparameter
        5001 = Autorouter: Dialog allgemeine Autorouter-Parameter
        5002 = Autorouter: Dialog Platzierungsparameter
        5003 = Autorouter: Dialog Autoroutingoptionen
        5004 = Autorouter: Dialog Autoroutingsteuerung
        5005 = Autorouter: Dialog Autoroutingstrategie
        5006 = Autorouter: Dialog Autoroutingbatcheinstellungen
        5000 = CAM-Prozessor: Dialog Bildarstellungsparameter
        5001 = CAM-Prozessor: Dialog Kontrollplotparameter
        5002 = CAM-Prozessor: Dialog Gerberplotparameter
        5003 = CAM-Prozessor: Dialog Bohrdatenausgabeparameter
        5004 = CAM-Prozessor: Dialog allgemeine CAM-/Plotparameter
        5000 = CAM-View: Dialog Bildarstellungsparameter
        5001 = CAM-View: Dialog allgemeine CAM-View-Parameter
```

Die Standardmenünummern werden mit der Formel

```
100 × Hauptmenünummer + Submenünummer
```

berechnet, wobei die Nummerierung der Menüpunkte jeweils bei Null beginnt. Die Hauptmenünummer 0 ist grunsätzlich für das Menü Undo, Redo reserviert, d.h. der Aufruf der im Menü Undo, Redo enthaltenen Funktionen ist nur über die hierfür speziell definierten Menünummern 9003 und 9004 möglich.

STD5 - Dialogelement Parametertyp:

```
                      Parameterwerte:
      0 = 0x000000 = String/Zeichenkette
      1 = 0x000001 = Bool'scher Wert (Checkbox)
      2 = 0x000002 = Integer-Wert
      3 = 0x000003 = Double-Wert
                      Anzeigeelementtypen:
      4 = 0x000004 = Beschriftung/Titel
      5 = 0x000005 = Horizontale Trennliniengrafik
      6 = 0x000006 = Vertikale Trennliniengrafik
                      Auswahlmenüelemente:
      7 = 0x000007 = Radioboxauswahl erste Option
      8 = 0x000008 = Radioboxauswahl nächste Option
      9 = 0x000009 = Selektionsmenü Basiseintrag
     10 = 0x00000A = Selektionsmenü nächster Eintrag
  65536 = 0x010000 = Listboxauswahlmenü Basiseintrag
  65537 = 0x010001 = Listboxauswahlmenü nächster Eintrag
                      Dialogschaltflächen:
     11 = 0x00000B = Aktions-Button
     12 = 0x00000C = OK-Button
     13 = 0x00000D = Abbruch-Button
                      Spezielle Parametertypkodierungen:
     14 = 0x00000E = Dummy-Dialogelement
 983055 = 0x0F000F = Parametertypmaske (für Parametertypabfragen)
                      Numerische Wertangaben:
     16 = 0x000010 = Vorzeichenbehafteter numerischer Wert
     32 = 0x000020 = Distanz- bzw. Längenangabe
     64 = 0x000040 = Drehwinkelangabe
                      Wertebereichsprüfung:
    128 = 0x000080 = Prüfparameter Wertebereichsminimum
    256 = 0x000100 = Prüfparameter Wertebereichsmaximum
    512 = 0x000200 = Unmittelbarer Prüfparameter Wertebereichsminimum
   1024 = 0x000400 = Unmittelbarer Prüfparameter Wertebereichsmaximum
                      Verschiedene Parametertypkodierungen:
   2048 = 0x000800 = Parametertyp Leerstring-Anzeigefeld
   4096 = 0x001000 = Parametertyp Zeichensatz Fettdruck
   8192 = 0x002000 = Parametertyp Zeichensatz feste Breite
  16384 = 0x004000 = Parametertyp Bestätigung durch Doppelklick
  32768 = 0x008000 = Parametertyp Editieren deaktiviert
1048576 = 0x100000 = Parametertyp Bestätigung durch Einfachklick
```

STD6 - Interaktionsmodus:

```
0 = Eingabe manuell
1 = Eingabe automatisch
```

STD7 - Koordinatenanzeigemodus:

```
0 = Anzeige/Eingabe in mm (im IC-Design in Mikrometer)
1 = Anzeige/Eingabe in Inch (im IC-Design in mm)
```

STD8 - Rasterfreigabe Flag:

```
0 = Raster freigeben
1 = Raster einhalten
```

STD9 - Winkelfreigabe Flag:

```
0 = Winkel freigeben
1 = Winkel einhalten
```

STD10 - Workspace Flag:

```
0 = Element/Objekt befindet sich nicht im Arbeitsbereich
1 = Element/Objekt befindet sich im Arbeitsbereich
```

STD11 - Fixiert Flag:

```
0 = Element/Objekt nicht fixiert
1 = Element/Objekt fixiert
```

STD12 - Elementverankerungsmodus:

```
0 = Element/Objekt ist nicht verankert
2 = Element/Objekt ist verankert
```

STD13 - Gruppen Flag:

```
0 = Element/Objekt nicht zur Gruppe gehörig
1 = Element/Objekt zur Gruppe gehörig
2 = Gruppenzugehörigkeit invertieren
```

STD14 - Spiegelungsmodus:

```
0 = Element/Objekt ist nicht gespiegelt
1 = Element/Objekt ist gespiegelt
```

STD15 - Polygonpunkttyp:

```
0 = Normaler Punkt
1 = Mittelpunkt Bogen links
2 = Mittelpunkt Bogen rechts
```

STD16 - Makrostatus:

```
xxxxxxx1 = Makro ist komplett geladen (Bitmaske)
xxxxxx1x = Makro fehlt (Bitmaske)
   sonst = intern
```

STD17 - Maustasten:

```
0 = Tastatureingabe
1 = Linke Maustaste
2 = Mittlere Maustaste
3 = Rechte Maustaste
```

STD18 - Farbwerte:

```
  0 = Schwarz (keine Farbe)
  1 = Blau
  2 = Grün
  3 = Kobaltblau
  4 = Rot
  5 = Violett
  6 = Braun
  7 = Hellgrau
  8 = Dunkelgrau
  9 = Hellblau
 10 = Hellgrün
 11 = Hellkobaltblau
 12 = Hellrot
 13 = Hellviolett
 14 = Gelb
 15 = Weiß
 -1 = Schwarz ausgeblendet
 -2 = Blau ausgeblendet
 -3 = Grün ausgblendet
 -4 = Kobaltblau ausgeblendet
 -5 = Rot ausgeblendet
 -6 = Violett ausgeblendet
 -7 = Braun ausgeblendet
 -8 = Hellgrau ausgeblendet
 -9 = Dunkelgrau ausgeblendet
-10 = Hellblau ausgeblendet
-11 = Hellgrün ausgeblendet
-12 = Hellkobaltblau ausgeblendet
-13 = Hellrot ausgeblendet
-14 = Hellviolett ausgeblendet
-15 = Gelb ausgeblendet
-16 = Weiß ausgeblendet
```

STD19 - Zeichenmodus:

```
0 = Ersetzen
1 = Löschen
2 = Setzen
3 = Komplement
```

STD20 - Polygonfüll-/zeichenmodus:

```
 0 = Geschlossenes Linienpolygon
 1 = Geschlossenes Füllpolygon
 2 = Offenes Linienpolygon
 3 = Geschlossenes Polygon mit Füllmuster
+4 = Gestrichelte Polygonumrandung
+8 = Gepunktete Polygonumrandung
```

STD21 - Interaktionspeichermodus:

```
0 = Interaktionsplatzhalter an das Ende der Interaktionswarteschlange anfügen
1 = Automatische Interaktion an das Ende der Interaktionswarteschlange anfügen
2 = Interaktionsplatzhalter am Beginn der Interaktionswarteschlange einfügen
3 = Automatische Interaktion am Beginn der Interaktionswarteschlange einfügen
```

A.2.2 Schematic Capture Wertebereiche (CAP)

Die nachfolgend aufgeführten Kodierungen besitzen Gültigkeit in den Definitionen für die Aufruftypen CAP und SCM. Sie definieren mithin gültige Wertebereiche für spezielle Elemente von Index-Variablen-Typen bzw. Systemfunktions-Parameter innerhalb der Interpreterumgebung des **Schaltplaneditor**. Die Benennung der Wertebereichsdefinitionen ergibt sich aus dem Schlüsselwort **CAP** und einer fortlaufenden Nummer.

CAP1 - Schaltplan Textmodus:

```
0 = Standardtext (als Bitmaske mit CAP7 kombinierbar)
1 = Kommentartext (als Bitmaske mit CAP7 kombinierbar)
```

CAP2 - Schaltplan Polygontyp:

```
0 = Grafiklinie
1 = Grafikfläche
2 = intern
3 = intern
4 = Kontaktbereich
5 = Punktlinie
```

CAP3 - Schaltplan Figurenelementtyp:

```
1 = Polygon
2 = Verbindung
3 = Benannte Referenz
4 = intern
5 = Text
6 = Namensmuster
7 = intern
8 = Polygon-Eck-/Pickpunkt
9 = Pick Benannte Symbolreferenz
10 = Pick Benannte Labelreferenz
11 = Attributpick Benannte Referenz
```

CAP4 - Schaltplan Poolelementtyp:

```
   -1 = Unbekanntes/ungültiges Element
    1 = Elementtyp Makro (C_MACRO)
    3 = Elementtyp Benannte Referenz (C_NREF)
    6 = Elementtyp Attributwert (C_ATTRIBUTE.VALUE)
    7 = Elementtyp Attributname (C_ATTRIBUTE.NAME)
   16 = Elementtyp Polygon (C_POLY)
   17 = Elementtyp Text (C_TEXT)
   18 = Elementtyp Verbindungssegmentliste (C_CONBASE)
   19 = Elementtyp Bustap (C_BUSTAP)
   20 = Elementtyp Bauteilnamensmuster (C_MACRO.PNAMEPAT)
   32 = Elementtyp Zeichensatzname
sonst = intern
```

CAP5 - Schaltplan Tagsymbol-/Taglabelmodus:

```
1 = Standardsymbol / Standardlabel
2 = Virtuelles Tagsymbol
3 = Netzlisten-Tagsymbol / Netzattributlabel
```

CAP6 - Schaltplan Tagpintyp:

```
0 = Standard Pin oder Label
1 = Symbolverweis
2 = Pinverweis
3 = Netzverweis
4 = Netzpinverweis
3 = Netzbereichsverweis
```

CAP7 - Schaltplan Textstil Bitmaske:

```
xxxx000x = Standardtextstil (kein Rahmen)
xxxxxx1x = Rahmen 1; Abstand zum Text 1/8 der Texthöhe
xxxxx1xx = Rahmen 2; Abstand zum Text 1/4 der Texthöhe
xxxx1xxx = Offene(r) Rahmen; ohne seitliche Rahmenlinie(n) am Textursprung
xxx1xxxx = Keine Textrotation
xx1xxxxx = Text horizontal zentriert
x1xxxxxx = Text vertikal zentriert
1xxxxxxx = Text rechts ausgerichtet
```

A.2.3 Schematic Editor Wertebereiche (SCM)

Die nachfolgend aufgeführten Kodierungen besitzen Gültigkeit in den Definitionen für den Aufruftyp SCM. Sie definieren mithin gültige Wertebereiche für spezielle Systemfunktions-Parameter innerhalb der Interpreterumgebung des **Schematic Editors**. Die Benennung der Wertebereichsdefinitionen ergibt sich aus dem Schlüsselwort `SCM` und einer fortlaufenden Nummer.

SCM1 - SCM Anzeigeelementtypen:

```
(-16384) = -0x4000 = ungültiges Anzeigeelement
       0 = Dokumentation
       1 = Verbindungen
       2 = Symbole
       3 = Marker
       4 = Symbolgrenzen
       5 = intern
       6 = intern
       7 = Anschlussfläche
       8 = Arbeitsbereich
       9 = Nullpunkt
      10 = Highlight
      11 = Kommentartext
      12 = Tagsymbol
      13 = Taglink
      14 = Variantenattribut
      15 = Plot unsichtbar (von Plotausgabe ausgenommene Elemente)
```

SCM2 - Schaltplan Anzeigeelementklassen Hierarchieebenen-Bitmaske:

```
xxxxxxxxxxx1 = Anzeige Element auf Planebene
xxxxxxxxxx1x = Anzeige Element auf Symbolebene
xxxxxxxxx1xx = Anzeige Element auf Labelebene
xxxxxxxx1xxx = Anzeige Element auf Markerebene
```

A.2.4 Layout Wertebereiche (LAY)

Die nachfolgend aufgeführten Kodierungen besitzen Gültigkeit in den Definitionen für die Aufruftypen LAY, GED, AR und CAM. Sie definieren mithin gültige Wertebereiche für spezielle Elemente von Index-Variablen-Typen bzw. Systemfunktions-Parameter innerhalb der Interpreterumgebungen des **Layouteditors**, des **Autorouters** und des **CAM-Prozessors**. Die Benennung der Wertebereichsdefinitionen ergibt sich aus dem Schlüsselwort LAY und einer fortlaufenden Nummer.

LAY1 - Layout Lagennummer:

```
(-16384) = -0x4000 = Ungültige Lage
    (-6) = Innenlagen
    (-5) = Oberste Lage
    (-4) = Airlines
    (-3) = Bestückungsplan
    (-2) = Umrandung
    (-1) = Alle Lagen
       0 = Signallage 1
       1 = Signallage 2
       2 = Signallage 3
       : = Signallage :
      99 = Signallage 100
     768 = 0x300 = Versorgungslage 1
     769 = 0x301 = Versorgungslage 2
     770 = 0x302 = Versorgungslage 3
       : =     : = Versorgungslage :
     777 = 0x309 = Versorgungslage 10
     778 = 0x30A = Versorgungslage 11
     779 = 0x30B = Versorgungslage 12
    1024 = 0x400 = Dokumentarlage 1 Seite 1
    1025 = 0x401 = Dokumentarlage 1 Seite 2
    1026 = 0x402 = Dokumentarlage 1 Beide Seiten
    1040 = 0x410 = Dokumentarlage 2 Seite 1
    1041 = 0x411 = Dokumentarlage 2 Seite 2
    1042 = 0x412 = Dokumentarlage 2 Beide Seiten
       : =     : = Dokumentarlage :
    1168 = 0x490 = Dokumentarlage 10 Seite 1
    1169 = 0x491 = Dokumentarlage 10 Seite 2
    1170 = 0x492 = Dokumentarlage 10 Beide Seiten
    1184 = 0x4A0 = Dokumentarlage 11 Seite 1
    1185 = 0x4A1 = Dokumentarlage 11 Seite 2
    1186 = 0x4A2 = Dokumentarlage 11 Beide Seiten
    1200 = 0x4B0 = Dokumentarlage 12 Seite 1
    1201 = 0x4B1 = Dokumentarlage 12 Seite 2
    1202 = 0x4B2 = Dokumentarlage 12 Beide Seiten
       : =     : = Dokumentarlage :
    2592 = 0xA20 = Dokumentarlage 99 Seite 1
    2593 = 0xA21 = Dokumentarlage 99 Seite 2
    2594 = 0xA22 = Dokumentarlage 99 Beide Seiten
    2608 = 0xA30 = Dokumentarlage 100 Seite 1
    2609 = 0xA31 = Dokumentarlage 100 Seite 2
    2610 = 0xA32 = Dokumentarlage 100 Beide Seiten
```

LAY2 - Layout Textmodus:

```
0 = Physical
1 = Logical
2 = Norotate
```

LAY3 - Layout Flächen Spiegelungsmodus:

```
0 = Immer sichtbar
1 = Ungespiegelt sichtbar
2 = Gespiegelt sichtbar
17 = Fixiert sichtbar wenn nicht gespiegelt
18 = Fixiert sichtbar wenn gespiegelt
```

LAY4 - Layout Polygontyp:

```
1 = Kupferfläche
2 = Sperrfläche
3 = Umrandung
4 = Potentialfläche
5 = Dokumentarlinie
6 = Dokumentarfläche
7 = Füllbereich
8 = Schraffierte Kupferfläche
9 = Split Power Plane Fläche
```

LAY5 - Layout Bohrungsklasse:

```
 0 = - (ungespiegelt Default)
 1 = A (Standard/ungespiegelt)
 2 = B (Standard/ungespiegelt)
 3 = C (Standard/ungespiegelt)
 4 = D (Standard/ungespiegelt)
 5 = E (Standard/ungespiegelt)
 6 = F (Standard/ungespiegelt)
 7 = G (Standard/ungespiegelt)
 8 = H (Standard/ungespiegelt)
 9 = I (Standard/ungespiegelt)
10 = J (Standard/ungespiegelt)
11 = K (Standard/ungespiegelt)
12 = L (Standard/ungespiegelt)
13 = M (Standard/ungespiegelt)
14 = N (Standard/ungespiegelt)
15 = O (Standard/ungespiegelt)
16 = P (Standard/ungespiegelt)
17 = Q (Standard/ungespiegelt)
18 = R (Standard/ungespiegelt)
19 = S (Standard/ungespiegelt)
20 = T (Standard/ungespiegelt)
21 = U (Standard/ungespiegelt)
22 = V (Standard/ungespiegelt)
23 = W (Standard/ungespiegelt)
24 = X (Standard/ungespiegelt)
25 = Y (Standard/ungespiegelt)
26 = Z (Standard/ungespiegelt)
0x0080 =  0 × 256 + 128 = - (gespiegelt Default)
0x0180 =  1 × 256 + 128 = A (gespiegelt)
0x0280 =  2 × 256 + 128 = B (gespiegelt)
0x0380 =  3 × 256 + 128 = C (gespiegelt)
0x0480 =  4 × 256 + 128 = D (gespiegelt)
0x0580 =  5 × 256 + 128 = E (gespiegelt)
0x0680 =  6 × 256 + 128 = F (gespiegelt)
0x0780 =  7 × 256 + 128 = G (gespiegelt)
0x0880 =  8 × 256 + 128 = H (gespiegelt)
0x0980 =  9 × 256 + 128 = I (gespiegelt)
0x0A80 = 10 × 256 + 128 = J (gespiegelt)
0x0B80 = 11 × 256 + 128 = K (gespiegelt)
0x0C80 = 12 × 256 + 128 = L (gespiegelt)
0x0D80 = 13 × 256 + 128 = M (gespiegelt)
0x0E80 = 14 × 256 + 128 = N (gespiegelt)
0x0F80 = 15 × 256 + 128 = O (gespiegelt)
0x1080 = 16 × 256 + 128 = P (gespiegelt)
0x1180 = 17 × 256 + 128 = Q (gespiegelt)
0x1280 = 18 × 256 + 128 = R (gespiegelt)
0x1380 = 19 × 256 + 128 = S (gespiegelt)
0x1480 = 20 × 256 + 128 = T (gespiegelt)
0x1580 = 21 × 256 + 128 = U (gespiegelt)
0x1680 = 22 × 256 + 128 = V (gespiegelt)
0x1780 = 23 × 256 + 128 = W (gespiegelt)
0x1880 = 24 × 256 + 128 = X (gespiegelt)
0x1980 = 25 × 256 + 128 = Y (gespiegelt)
0x1A80 = 26 × 256 + 128 = Z (gespiegelt)
```

Gespiegelte Bohrklassen können zur Definition gespiegelter partieller Durchkontaktierungen verwendet werden. Klassencodes für ungespiegelte und gespiegelte Bohrklassen können durch Addition bzw. bitweise Veroderung zu kombinierten Bohrklassenspezifikationen zusammengefasst werden, wie z.B. in

```
6 + 0x0880
```

für die Standardbohrklasse **F** (6) kombiniert mit der gespiegelten Bohrklasse **H** (0x0880).

LAY6 - Layout Figurenelementtyp:

```
1 = Polygon
2 = Bahn
3 = Benannte Referenz
4 = Namenlose Referenz
5 = Text
6 = Bohrung
7 = intern
8 = Polygon-Eck-/Pickpunkt
9 = Leiterbahn-Eck-/Pickpunkt
10 = Füllflächenpolygonpick
```

LAY7 - Layout Leveltyp:

```
>= 0 = Single Netz Level
(-1) = Multiple Netz Level (Kurzschluss)
(-2) = Geänderter Level, kein Netz
(-3) = zugewiesener Level (intern)
```

LAY8 - Layout Poolelementtyp:

```
   -1 = Unbekanntes/ungültiges Element
    1 = Elementtyp Makro (L_MACRO)
    5 = Elementtyp Unbenannte Referenz (L_UREF)
    6 = Elementtyp Benannte Referenz (L_NREF)
    8 = Elementtyp Attributwert (L_ATTRIBUTE.VALUE)
    9 = Elementtyp Attributname (L_ATTRIBUTE.NAME)
   16 = Elementtyp Polygon (L_POLY)
   17 = Elementtyp Leiterbahn (L_LINE)
   18 = Elementtyp Text (L_TEXT)
   19 = Elementtyp Bohrung (L_DRILL)
   21 = Elementtyp Leiterbahnschraffur-Polygon
   32 = Elementtyp Lage Bauteilseite
   33 = Elementtyp Versorgungslagen-Netz
   34 = Elementtyp DRC-Parameter
   35 = Elementtyp Zeichensatzname
   48 = Elementtyp DRC-Fehlermarker (L_DRCERROR)
sonst = intern
```

LAY9 - Layout Anzeigeelementtypen (zusätzlich zu LAY1):

```
 (-7) = Bohrungen (übertragen auf die Bohrklassen '-', 'A'-'Z')
 (-8) = Arbeitsbereich
 (-9) = Nullpunkt
(-10) = Fehler
(-11) = Highlight
(-12) = Bohrklasse '-'
(-13) = Bohrklasse 'A'
    : = :
(-38) = Bohrklasse 'Z'
(-39) = Fixiert
(-40) = Verankert
```

LAY10 - Layout Mincon Funktionstyp:

```
0 = Kein Mincon
1 = Pins horizontal
2 = Pins vertikal
3 = Pins horizontal+vertikal
4 = Pins Luftlinie
5 = Ecken horizontal
6 = Ecken vertikal
7 = Ecken horizontal+vertikal
8 = Ecken Luftlinie
```

LAY11 - Layout Eingabeinteraktionstyp:

```
>= 0 = Eingabe Poolelement
(-1) = Eingabe Gummiband
(-2) = Eingabe Fenster input
(-3) = Eingabe Kreismittelpunkt
(-4) = Eingabe Kreisbogen im (mathematisch positiven) Gegenuhrzeigersinn
(-5) = Eingabe Kreisbogen im (mathematisch negativen) Uhrzeigersinn
(-6) = Eingabe Segmentverschiebung
(-7) = Eingabe Segmentteilung
(-8) = Eingabe Segmentmarker
(-9) = Eingabe Gummiband Typ 2
```

LAY12 - Layout Variantensichtbarkeit:

```
0-99 = Visible for given variant number
100  = Visible for all variants
101  = Visible for unplaced variant
```

LAY13 - Layout DRC-Fehleranzeige:

```
1 = DRC Kupferabstandsverletzung
2 = DRC Dokumentarlagen-Sperrbereichsverletzung planar
3 = DRC Dokumentarlagen-Sperrbereichsverletzung vertical (Höhenverletzung)
4 = DRC HF-Designregelverletzung
5 = DRC ungültiger Bereich für Polygonablage
7 = DRC Bauteillagen-Entwurfsregelverletzung
8 = Füllpolygon Errorhinweis
|65536 = DRC Fehlermarker ausgeblendet
```

LAY14 - Layout Textstil Bitmaske:

```
xxx000xxxxxx = Standardtextstil (kein Rahmen)
xxxxx1xxxxxx = Rahmen 1; Abstand zum Text 1/8 der Texthöhe
xxxx1xxxxxxx = Rahmen 2; Abstand zum Text 1/4 der Texthöhe
xxx1xxxxxxxx = Offene(r) Rahmen; ohne seitliche Rahmenlinie(n) am Textursprung
xx1xxxxxxxxx = Text horizontal zentriert
x1xxxxxxxxxx = Text vertikal zentriert
1xxxxxxxxxxx = Text rechts ausgerichtet
```

LAY15 - Layout Elementanzeigemodus Bitmaske:

```
xxxxxxxxxxx1 = Elementanzeige auf Layoutebene
xxxxxxxxxx1x = Elementanzeige auf Bauteilebene
xxxxxxxxx1xx = Elementanzeige auf Padstackebene
xxxxxxxx1xxx = Elementanzeige auf Padebene
```

A.2.5 CAM-Prozessor Wertebereiche (CAM)

Die nachfolgend aufgeführten Kodierungen besitzen Gültigkeit in den Definitionen für den Aufruftyp CAM. Sie definieren mithin gültige Wertebereiche für spezielle Systemfunktions-Parameter innerhalb der Interpreterumgebung des **CAM-Prozessors**. Die Benennung der Wertebereichsdefinitionen ergibt sich aus dem Schlüsselwort "CAM" und einer fortlaufenden Nummer.

CAM1 - CAM-Prozessor Spiegelungsmodus:

```
0 = Spiegelung aus
1 = Spiegelung ein
2 = X-Rückseite (Spiegelung aus)
3 = X-Rückseite (Spiegelung ein)
4 = Y-Rückseite (Spiegelung aus)
5 = Y-Rückseite (Spiegelung ein)
```

CAM2 - CAM-Prozessor Gerberausgabe Länge einer Plottereinheit (STD2):

```
0.00002540000 = 2.3 Zoll Format
0.00000254000 = 2.4 Zoll Format
0.00000025400 = 2.5 Zoll Format
0.00000002540 = 2.6 Zoll Format
0.00000000254 = 2.7 Zoll Format
oder jeder andere Wert größer 0.00000000053
```

CAM3 - CAM-Prozessor Gerber-Ausgabeformat:

```
     0 = 2.3 Zölliges Format
     1 = 2.4 Zölliges Format
     2 = 2.5 Zölliges Format
     3 = 2.6 Zölliges Format
     4 = 3.3 Metrisches Format
     5 = 3.4 Metrisches Format
     6 = 3.5 Metrisches Format
     7 = 3.6 Metrisches Format
```

CAM4 - CAM-Prozessor HP-GL-Plot Stiftnummer:

```
     1 = Stift 1 aktiviert
     2 = Stift 2 aktiviert
     3 = Stift 3 aktiviert
     4 = Stift 4 aktiviert
     5 = Stift 5 aktiviert
     6 = Stift 6 aktiviert
     7 = Stift 7 aktiviert
     8 = Stift 8 aktiviert
     9 = Stift 9 aktiviert
    10 = Stift 10 aktiviert
     : = Stift  : aktiviert
    99 = Stift 99 aktiviert
  (-1) = Stift deaktiviert/ungültig
  (-2) = Stift   1 deaktiviert
  (-3) = Stift   2 deaktiviert
  (-4) = Stift   3 deaktiviert
  (-5) = Stift   4 deaktiviert
  (-6) = Stift   5 deaktiviert
  (-7) = Stift   6 deaktiviert
  (-8) = Stift   7 deaktiviert
  (-9) = Stift   8 deaktiviert
 (-10) = Stift   9 deaktiviert
 (-11) = Stift  10 deaktiviert
    :  = Stift   : deaktiviert
(-101) = Stift 100 deaktiviert
```

A.2.6 IC Design Wertebereiche (ICD)

Die nachfolgend aufgeführten Kodierungen besitzen Gültigkeit in den Definitionen für die Aufruftypen ICD und CED. Sie definieren mithin gültige Wertebereiche für spezielle Elemente von Index-Variablen-Typen bzw. Systemfunktions-Parameter innerhalb der Interpreterumgebung des **Chip Editors**. Die Benennung der Wertebereichsdefinitionen ergibt sich aus dem Schlüsselwort `ICD` und einer fortlaufenden Nummer.

ICD1 - IC Design Lagennummer:

```
(-16384) = -0x4000 = Ungültige Lage
    (-3) = Airlines
    (-2) = Umrandung
    (-1) = Alle Lagen
       0 = IC Lage 1
       1 = IC Lage 2
       2 = IC Lage 3
       : = IC Lage :
      99 = IC Lage 100
```

ICD2 - IC Design Textmodus:

```
0 = Physical
1 = Logical
2 = Norotate
```

ICD3 - IC Design Flächen Spiegelungsmodus:

```
0 = Immer sichtbar
1 = Ungespiegelt sichtbar
2 = Gespiegelt sichtbar
```

ICD4 - IC Design Polygontyp:

```
1 = Aktive Fläche
2 = Sperrfläche
3 = Dokumentarlinie
4 = Umrandung
```

ICD5 - IC Design Figurenelementtyp:

```
1 = Polygon
2 = Bahn
3 = Benannte Referenz
4 = Unbenannte Referenz
5 = Text
6 = intern
7 = Polygon-Eck-/Pickpunkt
8 = Leiterbahn-Eck-/Pickpunkt
```

ICD6 - IC Design Leveltyp:

```
>= 0 = Single Netz Level
(-1) = Multiple Netz Level (Kurzschluss)
(-2) = Geänderter Level, kein Netz
(-3) = zugewiesener Level (intern)
```

ICD7 - IC Design Poolelementtyp:

```
 (-1) = Unbekanntes/ungültiges Element
    1 = Elementtyp Makro (C_MACRO)
    2 = Elementtyp Unbenannte Referenz (C_UREF)
    3 = Elementtyp Benannte Referenz (C_NREF)
    6 = Elementtyp Attributwert (C_ATTRIBUTE.VALUE)
    7 = Elementtyp Attributname (C_ATTRIBUTE.NAME)
   16 = Elementtyp Polygon (C_POLY)
   17 = Elementtyp Leiterbahn (C_LINE)
   18 = Elementtyp Text (C_TEXT)
sonst = Internal
```

ICD8 - IC Design Anzeigeelementtypen (zusätzlich zu ICD1):

```
(-6) = Arbeitsbereich
(-7) = Nullpunkt
(-8) = Fehler
(-9) = Highlight
```

ICD9 - IC Design Lagenanzeigemodus Bitmuster:

```
xxxxx00 = Anzeige Lagenobjekte mit Randliniendarstellung
xxxxx01 = Anzeige Lagenobjekte mit Füllflächendarstellung
xxxxx10 = Anzeige Lagenobjekte mit Randliniendarstellung gestrichelt
xxxxx11 = Anzeige Lagenobjekte mit Füllflächenmuster
00000xx = Anzeige Lagenobjekte mit Füllflächenmuster 0
.....xx = Anzeige Lagenobjekte mit Füllflächenmuster :
11111xx = Anzeige Lagenobjekte mit Füllflächenmuster 31
```

ICD10 - IC Design Mincon Funktionstyp:

```
0 = Kein Mincon
1 = Pins horizontal
2 = Pins vertikal
3 = Pins horizontal+vertikal
4 = Pins Luftlinie
5 = Ecken horizontal
6 = Ecken vertikal
7 = Ecken horizontal+vertikal
8 = Ecken Luftlinie
```

Anhang B
Index-Variablen-Typen

Dieser Anhang beschreibt die in der **Bartels User Language** per Definition festgelegten Index-Variablen-Typen. Die Index-Variablen-Typen sind dabei gegliedert nach den jeweiligen Aufruftypen zunächst in Referenzlisten aufgeführt und anschließend in Form eines Nachschlagewerks in alphabetischer Reihenfolge beschrieben.

Inhalt

B.1 Index-Übersicht

Jeder in der **Bartels User Language** definierte Index-Variablen-Typ ist genau einem der Aufruftypen STD, CAP, LAY oder ICD zugeordnet. Dieser Abschnitt enthält eine Übersicht über die zum jeweiligen Aufruftyp definierten Index-Variablen-Typen.

B.1.1 Standard Index-Variablen-Typen (STD)

Die nachfolgend aufgelisteten Index-Variablen-Typen sind dem Aufruftyp STD zugeordnet, d.h. auf sie kann in allen Interpreterumgebungen des **Bartels AutoEngineer** zugegriffen werden:

BAEPARAM	Bartels AutoEngineer Parameter
GLOBALVAR	Globale User Language Variable

B.1.2 Schematic Capture Index-Variablen-Typen (CAP)

Die nachfolgend aufgelisteten Index-Variablen-Typen sind dem Aufruftyp CAP zugeordnet, d.h. auf sie kann in der Interpreterumgebung **Schaltplaneditor** zugegriffen werden:

C_ATTRIBUTE	Schaltplan Attribut
C_BUSTAP	Schaltplan Busanschluss
C_CNET	Schaltplan Logische Netzliste
C_CONBASE	Schaltplan Verbindungssegmentgruppe
C_CONSEG	Schaltplan Verbindungssegment
C_FIGURE	Schaltplan Figurenelement
C_LEVEL	Schaltplan Signalpotential
C_MACRO	Schaltplan Bibliothekselement
C_NREF	Schaltplan Makroreferenz (benannt)
C_POINT	Schaltplan Polygonpunkt
C_POLY	Schaltplan Polygon
C_POOL	Schaltplan Poolelement
C_TEXT	Schaltplan Text
CL_ALTPLNAME	Layoutnetzlisteneintrag Alternativbauform
CL_ATTRIBUTE	Layoutnetzlisteneintrag Attribut
CL_CNET	Layoutnetzliste
CL_CPART	Layoutnetzlisteneintrag Bauteil
CL_CPIN	Layoutnetzlisteneintrag Bauteilpin

B.1.3 Layout Index-Variablen-Typen (LAY)

Die nachfolgend aufgelisteten Index-Variablen-Typen sind dem Aufruftyp LAY zugeordnet, d.h. auf sie kann in den Interpreterumgebungen **Layouteditor**, **Autorouter** und **CAM-Prozessor** zugegriffen werden:

L_ALTPLNAME	Layoutnetzlisteneintrag Alternativbauform
L_ATTRIBUTE	Layoutnetzlisteneintrag Attribut
L_CNET	Layoutnetzliste
L_CPART	Layoutnetzlisteneintrag Bauteil
L_CPIN	Layoutnetzlisteneintrag Bauteilpin
L_DRCERROR	Layout DRC-Fehlermarker
L_DRCERROROK	Layout DRC-Fehlerakzeptierung
L_DRILL	Layout Bohrung
L_FIGURE	Layout Figurenelement
L_LEVEL	Layout Signalpotential
L_LINE	Layout Leiterbahn
L_MACRO	Layout Bibliothekselement
L_NREF	Layout Makroreferenz (benannt)
L_POINT	Layout Polygonpunkt
L_POLY	Layout Polygon
L_POOL	Layout Poolelement
L_POWLAYER	Layout Versorgungslage
L_TEXT	Layout Text
L_UREF	Layout Makroreferenz (unbenannt)

B.1.4 CAM-View Index-Variablen-Typen (CV)

Die nachfolgend aufgelisteten Index-Variablen-Typen sind dem Aufruftyp CV zugeordnet, d.h. auf sie kann in der Interpreterumgebung **CAM-View** zugegriffen werden:

CV_DATASET	CAM-View Datensatz

B.1.5 IC Design Index-Variablen-Typen (ICD)

Die nachfolgend aufgelisteten Index-Variablen-Typen sind dem Aufruftyp ICD zugeordnet, d.h. auf sie kann in der Interpreterumgebung **Chipeditor** zugegriffen werden:

I_ATTRIBUTE	IC-Design Netzlistentrag Attribut
I_CNET	IC-Design Netzliste
I_CPART	IC-Design Netzlisteneintrag Bauteil
I_CPIN	IC-Design Netzlisteneintrag Bauteilpin
I_FIGURE	IC Design Figurenelement
I_LEVEL	IC Design Signalpotential
I_LINE	IC Design Leiterbahn
I_MACRO	IC Design Bibliothekselement
I_NREF	IC Design Makroreferenz (benannt)
I_POINT	IC Design Polygonpunkt
I_POLY	IC Design Polygon
I_POOL	IC Design Poolelement
I_TEXT	IC Design Text
I_UREF	IC Design Makroreferenz (unbenannt)

B.2 Standard Index-Beschreibung (STD)

Dieser Abschnitt beschreibt die in der **Bartels User Language** definierten Index-Variablen-Typen für den allgemeinen Datenzugriff (STD).

BAEPARAM - Bartels AutoEngineer Parameter

Der Index **BAEPARAM** ermöglicht den Zugriff auf die aktuell definierten BAE-Parameter und deren Werte. Die Strukturdefinition von **BAEPARAM** lautet:

```
index BAEPARAM {              // BAE Parameter Index
      int IDCODE;             // Parameter Typ/Ident. Code
      int TYP;                // Parameter Datentyp:
                              //    1 = int
                              //    2 = double
                              //    4 = string
      int VALINT;             // Parameter Integer-Wert (für TYP 1)
      double VALDBL;          // Parameter Double-Wert (für TYP 2)
      string VALSTR;          // Parameter String-Wert (für TYP 4)
      };
```

Der Quellcode der **User Language**-Includedatei **baeparam.ulh** enthält die Liste der gültigen BAE-Parametertypen (d.h. die möglichen Werte für **IDCODE**) sowie Template-Funktionen für die Abfrage spezieller Parameterwerte.

GLOBALVAR - Globale User Language Variable

Der Index **GLOBALVAR** ermöglicht den Zugriff auf die globalen Variablen, die in der aktuellen Interpreterumgebung mit Hilfe der Systemfunktion **varset** aktuell definiert sind. Die Strukturdefinition von **GLOBALVAR** lautet:

```
index GLOBALVAR {             // Globale User Language Variable Index
      string NAME;            // Globale Variable Name
      int TYP;                // Globale Variable Datentyp:
                              //    1 = int
                              //    2 = double
                              //    3 = char
                              //    4 = string
      };
```

Der Wert einer mit dem Index **GLOBALVAR** gescannten globalen Variablen kann mit Hilfe der Systemfunktion **varget** abgefragt werden.

B.3 Schematic Capture Index-Beschreibung (CAP)

Dieser Abschnitt beschreibt die in der **Bartels User Language** definierten Index-Variablen-Typen für den Schematic Capture Datenzugriff (CAP).

C_ATTRIBUTE - Schaltplan Attribut

Über den Index **C_ATTRIBUTE** ist der Zugriff auf die Bauteil-Attribute im Schaltplan möglich. Die Strukturdefinition von **C_ATTRIBUTE** lautet:

```
index C_ATTRIBUTE {             // Attribut Index
      string NAME;              // Attribut Name
      string VALUE;             // Attribut Wert
      };
```

Der Index **C_ATTRIBUTE** ist nur als **of**-Index für die Liste der Attribute in **C_NREF** zu verwenden.

C_BUSTAP - Schaltplan Busanschluss

Über den Index **C_BUSTAP** ist der Zugriff auf die Busanschlüsse (Bustaps) des aktuell geladenen Schaltplans möglich. Die Strukturdefinition von **C_BUSTAP** lautet:

```
index C_BUSTAP {                // Busanschluss Index
      string NAME;              // Busanschluss Name
      double X;                 // Busanschluss X-Koordinate (STD2)
      double Y;                 // Busanschluss Y-Koordinate (STD2)
      double ANGLE;             // Busanschluss Drehwinkel (STD3)
      int MIRROR;               // Busanschluss Spiegelung (STD14)
      index C_MACRO MACRO;      // Verweis auf Busanschluss Makro
      index C_CONBASE CON;      // Verweis auf Verbindungssegmentgruppe
      index C_CONSEG SEG;       // Verweis auf Verbindungssegment
      };
```

Die Koordinaten des Busanschlusses liegen immer auf einem Verbindungssegment des entsprechenden Busses. Der Drehwinkel des Busanschlusses ist grundsätzlich ein Vielfaches von 90 Grad. Die Indexvariable **CON** ermöglicht den Rückwärtsverweis auf die Verbindungssegmentgruppe, zu der der entsprechende Busanschluss gehört. Die Indexvariable **SEG** ermöglicht den Querverweis auf das Verbindungssegment des Busses, auf dem der Busanschluss platziert ist.

C_CNET - Schaltplan Logische Netzliste

Über den Index **C_CNET** ist der Zugriff auf die logische Netzliste des aktuell geladenen Schaltplans möglich. Die Strukturdefinition von **C_CNET** lautet:

```
index C_CNET {                  // Logische Netzliste Index
      string NAME;              // Netz Name
      int NUMBER;               // Netz Nummer
      int NETGLO;               // Netz Globalkennzeichnung:
                                //    0 = Lokale Netzdefinition
                                //    1 = Globale Netzdefinition
      int BUSNET;               // Busnetznummer
      };
```

C_CONBASE - Schaltplan Verbindungssegmentgruppe

Über den Index **C_CONBASE** ist der Zugriff auf die Gruppe der Verbindungssegmente einer bestimmten Zeile bzw. Spalte des aktuell geladenen Schaltplans möglich. Die Strukturdefinition von **C_CONBASE** lautet:

```
index C_CONBASE {              // Verbindungssegmentgruppe Index
      int ORI;                 // Orientierung:
                               //    0 = Horizontal
                               //    1 = Vertikal
      double X;                // Basis-X-Koordinate (STD2)
      double Y;                // Basis-Y-Koordinate (STD2)
      int SN;                  // Anzahl Verbindungssegmente
      index C_CONSEG;          // Liste der Verbindungssegmente
      index C_BUSTAP;          // Liste der Busanschlüsse
      };
```

Über einen **C_CONBASE** Index sind alle Verbindungssegmente einer bestimmten Spalte (Orientierung vertikal) bzw. Reihe (Orientierung horizontal) einschließlich der daran definierten Busanschlüsse zusammengefasst. Der Zugriff auf die zur Verbindungssegmentgruppe gehörenden Verbindungssegmente bzw. Busanschlüsse erfolgt über entsprechende **forall-of**-Schleifen.

C_CONSEG - Schaltplan Verbindungssegment

Über den Index **C_CONSEG** ist der Zugriff auf die Verbindungssegmente des aktuell geladenen Schaltplans möglich. Die Strukturdefinition von **C_CONSEG** lautet:

```
index C_CONSEG {               // Verbindungssegment Index
      double X1;               // Segment X-Koordinate 1 (STD2)
      double Y1;               // Segment Y-Koordinate 1 (STD2)
      double X2;               // Segment X-Koordinate 2 (STD2)
      double Y2;               // Segment Y-Koordinate 2 (STD2)
      int BUSFLAG;             // Segment Buskennzeichnung:
                               //    0 = Normales Segment
                               //    1 = Bussegment
      int GROUP;               // Segment Gruppenflag (STD13)
      index C_CONBASE CON;     // Verweis auf Segmentgruppe
      };
```

Verbindungssegmente sind immer orthogonal platziert, d.h. entweder die X-Koordinaten sind identisch (Orientierung vertikal), oder die Y-Koordinaten sind identisch (Orientierung horizontal). Die Indexvariable **CON** ermöglicht den Rückwärtsverweis auf die Segmentgruppe, zu der das entsprechende Verbindungssegment gehört.

C_FIGURE - Schaltplan Figurenelement

Über den Index **C_FIGURE** ist der Zugriff auf die platzierten Figurenelemente (Polygone, Verbindungen, Makroreferenzen, Texte) des aktuell geladenen Elements möglich. Die Strukturdefinition von **C_FIGURE** lautet:

```
index C_FIGURE {              // Figurenelement Index
      int TYP;                // Element Typ (CAP3)
      string NAME;            // Element Name
      double SIZE;            // Element Größe (STD2)
      double X;               // Element X-Koordinate (STD2)
      double Y;               // Element Y-Koordinate (STD2)
      double ANGLE;           // Element Drehwinkel (STD3)
      int MIRROR;             // Element Spiegelung (STD14)
      int GROUP;              // Element Gruppenflag (STD13)
      index C_POOL POOL;      // Verweis auf Poolelement
      index C_POLY POLY;      // Verweis auf Polygonelement
      index C_CONBASE CONBASE;// Verweis auf Verbindungssegmentgruppe
      index C_NREF NREF;      // Verweis auf Makroreferenz (benannt)
      index C_TEXT TEXT;      // Verweis auf Textelement
      };
```

Die Indexvariable **NAME** gibt für benannte Makroreferenzen den Namen und für Texte den Textstring an. Auf SCM-Symbolebene enthält **NAME** das aktuell definierte Bauteilnamensmuster. Die Indexvariable **POOL** ermöglicht den Zugriff auf das Bibliotheks- bzw. Poolelement, aus dem das Figurenelement aufgebaut ist. Einzelne Daten eines Figurenelements können mit den Funktionen **scm_elem*chg** modifiziert werden. Der komplette Datensatz eines Figurenelements kann mit der Funktion **cap_scanfelem** über alle Hierarchieebenen abgearbeitet werden.

C_LEVEL - Schaltplan Signalpotential

Über den Index **C_LEVEL** ist der Zugriff auf die Connectivity-Level, d.h. auf die Netzlisten- bzw. Signalpotentiale des aktuell geladenen Schaltplans möglich. Die Strukturdefinition von **C_LEVEL** lautet:

```
index C_LEVEL {               // Connectivity-Level Index
      int IDNUM;              // Level Identifikationsnummer
      int BUSFLAG;            // Level Buskennzeichnung
      int SEGFLAG;            // Level Segmentverbindungsmodus (Bitmuster):
                              //    1 = Segment verbunden mit Level
                              //    2 = Kontaktflächen verbunden mit Level
      int ERRFLAG;            // Level Fehlerkennzeichnung
      int HIGHLIGHT;          // Level Highlight Flag
      int DISPLAY;            // Level Anzeige Attribute
      int CNN;                // Level Netzanzahl
      index C_CNET;           // Liste der Netze
      };
```

Der Zugriff auf die zum Signalpotential gehörenden Netze erfolgt über eine entsprechende **`forall-of`**-Schleife.

C_MACRO - Schaltplan Bibliothekselement

Über den Index **C_MACRO** ist der Zugriff auf die Makros, d.h. auf die Bibliothekselemente (Symbole, Labels, Marker) des aktuell geladenen Elements möglich. Die Strukturdefinition von **C_MACRO** lautet:

```
index C_MACRO {               // Makrodefinition Index
      string NAME;            // Makro Name
      double MLX;             // Linke Makrobegrenzung (STD2)
      double MLY;             // Untere Makrobegrenzung (STD2)
      double MUX;             // Rechte Makrobegrenzung (STD2)
      double MUY;             // Obere Makrobegrenzung (STD2)
      double MNX;             // Nullpunkt X-Koordinate (STD2)
      double MNY;             // Nullpunkt Y-Koordinate (STD2)
      int CLASS;              // Makro Klasse (STD1)
      int TAGSYM;             // Makro Tagsymbol-/Taglabelmodus (CAP5)
      int COMP;               // Makro Status (STD16)
      string PNAMEPAT;        // Makro Bauteilnamensmuster
      };
```

Über die Indexvariable **`PNAMEPAT`** kann das Bauteilnamensmuster für Stromlaufsymbole ermittelt werden.

C_NREF - Schaltplan Makroreferenz (benannt)

Über den Index **C_NREF** ist der Zugriff auf die benannten Makroreferenzen, d.h. auf die auf dem aktuell geladenen Element namentlich platzierten Bibliothekselemente möglich. Auf Planebene sind dies die platzierten Bauteile und Labels; auf Symbol- bzw. Labelebene sind dies die platzierten Pins. Die Strukturdefinition von **C_NREF** lautet:

```
index C_NREF {                // Benannte Referenz Index
      string NAME;            // Referenz Name
      double X;               // Referenz X-Koordinate (STD2)
      double Y;               // Referenz Y-Koordinate (STD2)
      double ANGLE;           // Referenz Drehwinkel (STD3)
      int MIRROR;             // Referenz Spiegelung (STD14)
      int TAGPTYP;            // Referenz Tagpintyp (CAP6)
      index C_MACRO MACRO;    // Verweis auf Makro
      index C_ATTRIBUTE;      // Liste der Attribute
      };
```

Über die Indexvariable **`MACRO`** ist der Zugriff auf das durch das **C_NREF**-Element referenzierte Bibliothekselement möglich. Der Zugriff auf die am Bauteil definierten Attribute erfolgt über eine entsprechende **`forall-of`**-Schleife.

C_POINT - Schaltplan Polygonpunkt

Über den Index **C_POINT** ist der Zugriff auf einzelne Polygonpunkte eines Polygons möglich. Die Strukturdefinition von **C_POINT** lautet:

```
index C_POINT {               // Polygonpunkt Index
      double X;               // Polygonpunkt X-Koordinate (STD2)
      double Y;               // Polygonpunkt Y-Koordinate (STD2)
      int TYP;                // Polygonpunkt Typ (STD15)
      };
```

Der Index **C_POINT** ist nur als **`of`**-Index für die Liste der Polygonpunkte in **C_POLY** zu verwenden.

C_POLY - Schaltplan Polygon

Über den Index **C_POLY** ist der Zugriff auf die im aktuell geladenen Element definierten Polygone (Flächen, Linien) möglich. Die Strukturdefinition von **C_POLY** lautet:

```
index C_POLY {                // Polygon Index
      int TYP;                // Polygon Typ (CAP2)
      double WIDTH;           // Polygon Linienbreite (STD2)
      double DASHLEN;         // Polygon Strichelungslänge (STD2)
      double DASHSPC;         // Polygon Strichelungsrelativabstand
      int DASH;               // Polygon Strichelungsmodus
      int PN;                 // Anzahl Polygonpunkte
      index C_POINT;          // Liste der Polygonpunkte
      };
```

Der Zugriff auf die zum Polygon gehörenden Polygonpunkte erfolgt über eine entsprechende **`forall-of`**-Schleife.

C_POOL - Schaltplan Poolelement

Über den Index **C_POOL** ist der Zugriff auf die aktuell geladenen Poolelemente möglich. Die Strukturdefinition von **C_POOL** lautet:

```
index C_POOL {                // Poolelement Index
      int TYP;                // Poolelement Typ (CAP4)
      int REFCNT;             // Poolelement Referenzierungsanzahl
      index C_POOL NXT;       // Verweis auf nächstes Poolelement
      index C_POOL REF;       // Verweis auf Referenz-Poolelement
      index C_POLY POLY;      // Verweis auf Polygonelement
      index C_CONBASE CONBASE;// Verweis auf Verbindungssegmentgruppe
      index C_NREF NREF;      // Verweis auf Makroreferenz (benannt)
      index C_TEXT TEXT;      // Verweis auf Textelement
      index C_MACRO MACRO;    // Verweis auf Bibliothekselement
      index C_BUSTAP BUSTAP;  // Verweis auf Busanschluss
      };
```

Der Index **C_POOL** wird benötigt zur Abarbeitung von Bibliotheksdefinitionen mit Hilfe der Systemfunktion **cap_scanpool**. Die Indexvariable **`REFCNT`** gibt an, wie oft das entsprechende Poolelement innerhalb des aktuell geladenen Elements referenziert wird. Die Indexvariablen **`NXT`** und **`REF`** dienen der schnellen Abarbeitung der aktuell referenzierten Poolelemente.

C_TEXT - Schaltplan Text

Über den Index **C_TEXT** ist der Zugriff auf die im aktuell geladenen Element definierten Texte möglich. Die Strukturdefinition von **C_TEXT** lautet:

```
index C_TEXT {                // Text Index
      string STR;             // Text String
      double X;               // Text X-Koordinate (STD2)
      double Y;               // Text Y-Koordinate (STD2)
      double ANGLE;           // Text Drehwinkel (STD3)
      double SIZE;            // Text Größe (STD2)
      double WIDTH;           // Text Linienbreite (STD2)
      int MIRROR;             // Text Spiegelung (STD14)
      int MODE;               // Text Modus/Stil (CAP1|CAP7)
      int CLASS;              // Text Klassenbitmuster
      };
```

CL_ALTPLNAME - Layoutnetzlisteneintrag Alternativbauform

Über den Index **CL_ALTPLNAME** ist der Zugriff auf die Liste der Alternativbauformen der aktuell geladenen Layoutnetzliste möglich. Die Strukturdefinition von **CL_ALTPLNAME** lautet:

```
index CL_ALTPLNAME {            // Layoutnetzliste Alternativbauformindex
      string PLNAME;            // Alternativbauform Name
      };
```

CL_ATTRIBUTE - Layoutnetzlisteneintrag Attribut

Über den Index **CL_ATTRIBUTE** ist der Zugriff auf die Bauteil- und Netzattribute der aktuell geladenen Layoutnetzliste möglich. Die Strukturdefinition von **CL_ATTRIBUTE** lautet:

```
index CL_ATTRIBUTE {            // Layoutnetzliste Attributindex
      string NAME;              // Attribut Name
      string VALUE;             // Attribut Wert
      };
```

CL_CNET - Layoutnetzliste

Über den Index **CL_CNET** ist der Zugriff auf die Netze der aktuell geladenen Layoutnetzliste möglich. Die Strukturdefinition von **CL_CNET** lautet:

```
index CL_CNET {                 // Layoutnetzliste Netzindex
      string NAME;              // Netz Name
      int NUMBER;               // Netz Nummer
      int PRIOR;                // Netz Routingpriorität
      double RDIST;             // Netz Mindestabstand (STD2)
      int PINN;                 // Anzahl Pins
      index CL_CPIN;            // Liste der Pins
      index CL_ATTRIBUTE;       // Liste der Attribute
      };
```

Die Netznummer wird allgemein zur Identifizierung eines Netzes herangezogen. Mit der Funktion **cap_getlaytreeidx** kann ausgehend von der angegebenen Netznummer der entsprechende **CL_CNET** Index gefunden werden. Der Mindestabstand gibt die Distanz an, die Leiterbahnen dieses Netzes zu nicht dem entsprechenden Netz zugehörenden Kupferstrukturen mindestens einhalten müssen. Der Zugriff auf die zum Netz gehörenden Pins bzw. Attribute erfolgt über eine entsprechende `forall-of`-Schleife.

CL_CPART - Layoutnetzlisteneintrag Bauteil

Über den Index **CL_CPART** ist der Zugriff auf die Bauteile der aktuell geladenen Layoutnetzliste möglich. Die Strukturdefinition von **CL_CPART** lautet:

```
index CL_CPART {                // Layoutnetzliste Bauteilindex
      string NAME;              // Bauteil Name
      string PLNAME;            // Physikalisches Bibliotheksteil
      int PEQUC;                // Bauteil Äquivalenz-Code
      int PINN;                 // Anzahl Pins
      int FPINN;                // Anzahl nicht angeschlossene Pins
      index CL_CPIN;            // Liste der Pins
      index CL_ALTPLNAME;       // Liste der Alternativbauformen
      index CL_ATTRIBUTE;       // Liste der Attribute
      };
```

Auf die Pinliste, die Alternativbauformen und die Attributwerte kann mit Hilfe von entsprechenden `forall-of` Schleifen zugegriffen werden. Bauteile mit identischen Äquivalenz-Codes dürfen auf ihren Einbauplätzen im Zuge einer Platzierungsoptimierung vertauscht werden (Component Swap).

CL_CPIN - Layoutnetzlisteneintrag Bauteilpin

Über den Index **CL_CPIN** ist der Zugriff auf die Bauteilpins der aktuell geladenen Layoutnetzliste möglich. Die Strukturdefinition von **CL_CPIN** lautet:

```
index CL_CPIN {                // Layoutnetzliste Bauteilpinindex
      string NAME;             // Pin Name
      double RWIDTH;           // Pin Routingbreite (STD2)
      int TREE;                // Pin Netznummer
      int GATE;                // Pin Gatternummer
      int GEQUC;               // Pin Gatter-Äquivalenz-Code
      int GEQUP;               // Pin Äquivalenz-Code
      int GGRPC;               // Pin Gatter-Gruppennummer
      int GPNUM;               // Pin Gatter-Relativnummer
      index CL_CNET CNET;      // Verweis auf Netz
      index CL_CPART CPART;    // Verweis auf Bauteil
      index CL_ATTRIBUTE;      // Liste der Pinattribute
      };
```

Die Pin Routingbreite gibt die Breite an, mit der vom Pin zum nächsten Verbindungspunkt geroutet werden soll. Über die Indexvariablen **`CNET`** bzw. **`CPART`** ist der Rückwärtsverweis auf das Netz bzw. das Bauteil, an dem der Pin definiert ist, möglich. Über die Indexvariablen **`GATE`**, **`GEQUC`**, **`GEQUP`**, **`GGRPC`** und **`GPNUM`** kann die Zulässigkeit der Gatter- und Pinvertauschbarkeit (Pin/Gate Swap) ermittelt werden.

B.4 Layout Index-Beschreibung (LAY)

Dieser Abschnitt beschreibt die in der **Bartels User Language** definierten Index-Variablen-Typen für den Layout Datenzugriff (LAY).

L_ALTPLNAME - Layoutnetzlisteneintrag Alternativbauform

Über den Index **L_ALTPLNAME** ist der Zugriff auf die in der Netzliste des aktuell geladenen Layouts eingetragenen Alternativbauformen möglich. Die Strukturdefinition von **L_ALTPLNAME** lautet:

```
index L_ALTPLNAME {             // Alternativbauform Index
      string PLNAME;            // Alternativbauform Name
      };
```

L_ATTRIBUTE - Layoutnetzlisteneintrag Attribut

Über den Index **L_ATTRIBUTE** ist der Zugriff auf die in der Netzliste des aktuell geladenen Layouts eingetragenen Bauteil- bzw. Netzattribute möglich. Die Strukturdefinition von **L_ATTRIBUTE** lautet:

```
index L_ATTRIBUTE {             // Attribut Index
      string NAME;              // Attribut Name
      string VALUE;             // Attribut Wert
      };
```

L_CNET - Layoutnetzliste

Über den Index **L_CNET** ist der Zugriff auf die in der Netzliste des aktuell geladenen Layouts eingetragenen Netze möglich. Die Strukturdefinition von **L_CNET** lautet:

```
index L_CNET {                  // Layoutnetzliste
      string NAME;              // Netz Name
      int NUMBER;               // Netz Nummer
      int PRIOR;                // Netz Routingpriorität
      double RDIST;             // Netz Mindestabstand (STD2)
      int VIS;                  // Netz Sichtbarkeitsflag
      int PINN;                 // Anzahl Pins
      index L_CPIN;             // Liste der Pins
      index L_ATTRIBUTE;        // Liste der Attribute
      index L_POOL UNRPOOL;     // Verweis auf Unroutes Poolelement
      };
```

Die Netznummer wird allgemein zur Identifizierung eines Netzes herangezogen. Mit der Funktion **lay_gettreeidx** kann ausgehend von der angegebenen Netznummer der entsprechende **L_CNET** Index gefunden werden. Der Mindestabstand gibt die Distanz an, die Leiterbahnen dieses Netzes zu nicht dem entsprechenden Netz zugehörenden Kupferstrukturen mindestens einhalten müssen. Der Zugriff auf die zum Netz gehörenden Pins bzw. Attribute erfolgt über eine entsprechende **`forall-of`**-Schleife. Die Indexvariable **`UNRPOOL`** ermöglicht den Zugriff auf die noch nicht verlegten Verbindungen (Unroutes, Airlines) des entsprechenden Netzes; die Abarbeitung dieser Verbindungen kann mit Hilfe der Funktion **lay_scanpool** erfolgen.

L_CPART - Layoutnetzlisteneintrag Bauteil

Über den Index **L_CPART** ist der Zugriff auf die in der Netzliste des aktuell geladenen Layouts eingetragenen Bauteile möglich. Die Strukturdefinition von **L_CPART** lautet:

```
index L_CPART {              // Netzliste Bauteilindex
      string NAME;           // Bauteil Name
      string PLNAME;         // Physikalisches Bibliotheksteil
      int USED;              // Bauteilplatzierungscode:
                             //    0 = Bauteil nicht platziert
                             //    1 = Bauteil platziert
                             //    2 = Bauteil platziert und zur Gruppe
selektiert
      int PEQUC;             // Bauteil Äquivalenz-Code
      int PINN;              // Anzahl Pins
      int FPINN;             // Anzahl nicht angeschlossene Pins
      index L_MACRO MACRO;   // Verweis auf Makro
      index L_CPIN;          // Liste der Pins
      index L_ALTPLNAME;     // Liste der Alternativbauformen
      index L_ATTRIBUTE;     // Liste der Attribute
      };
```

Auf die Pinliste, die Alternativbauformen und die Attributwerte kann mit Hilfe von entsprechenden **`forall-of`** Schleifen zugegriffen werden. Bauteile mit identischen Äquivalenz-Codes dürfen auf ihren Einbauplätzen im Zuge einer Platzierungsoptimierung vertauscht werden (Component Swap).

L_CPIN - Layoutnetzlisteneintrag Bauteilpin

Über den Index **L_CPIN** ist der Zugriff auf die in der Netzliste des aktuell geladenen Layouts eingetragenen Bauteilpins möglich. Die Strukturdefinition von **L_CPIN** lautet:

```
index L_CPIN {               // Netzliste Pin Index
      string NAME;           // Pin Name
      double RWIDTH;         // Pin Routingbreite (STD2)
      int TREE;              // Pin Netznummer
      int GATE;              // Pin Gatternummer
      int GEQUC;             // Pin Gatter-Äquivalenz-Code
      int GEQUP;             // Pin Äquivalenz-Code
      int GGRPC;             // Pin Gatter-Gruppennummer
      int GPNUM;             // Pin Gatter-Relativnummer
      index L_CNET CNET;     // Verweis auf Netz
      index L_CPART CPART;   // Verweis auf Bauteil
      };
```

Die Pin Routingbreite gibt die Breite an, mit der von dem Pin zum nächsten Verbindungspunkt geroutet werden soll. Über die Indexvariablen **`CNET`** bzw. **`CPART`** ist der Rückwärtsverweis auf das Netz bzw. das Bauteil, an dem der Pin definiert ist, möglich. Über die Indexvariablen **`GATE`**, **`GEQUC`**, **`GEQUP`**, **`GGRPC`** und **`GPNUM`** kann die Zulässigkeit der Gatter- und Pinvertauschbarkeit (Pin/Gate Swap) ermittelt werden.

L_DRCERROR - Layout DRC-Fehlermarker

Über den Index **L_DRCERROR** ist der Zugriff auf die vom Design Rule Check angezeigten Fehlermarker des aktuell geladenen Layoutelements möglich. Die Strukturdefinition von **L_DRCERROR** lautet:

```
index L_DRCERROR {              // DRC-Fehlermarker Index
      int TYP;                  // DRC-Fehlertyp:
                                //    1 = Kupferabstandsverletzung
                                //    2 = Dokumentarlagen-Sperrflächenverletzung
                                //    3 = Dokumentarlagen-
                                //         Höhensperrflächenverletzung
                                //    4 = HF-Designregelverletzung
                                //    5 = Polygonablage-Designregelverletzung
      int LAYER;                // DRC-Fehlermarker Lage (LAY1)
      double RLX;               // DRC-Fehlermarker linke Begrenzung (STD2)
      double RLY;               // DRC-Fehlermarker untere Begrenzung (STD2)
      double RUX;               // DRC-Fehlermarker rechte Begrenzung (STD2)
      double RUY;               // DRC-Fehlermarker obere Begrenzung (STD2)
      double CHKDIST;           // DRC-Fehlermarker DRC-Mindestabstand (STD2)
      double ERRDIST;           // DRC-Fehlermarker tatsächlicher
                                /. Elementabstand (STD2)
      string IDSTR;             // DRC-Fehler-Id-String
      index L_FIGURE FIG1;      // DRC-Fehlerelement 1
      index L_FIGURE FIG2;      // DRC-Fehlerelement 2
      };
```

L_DRCERROROK - Layout DRC-Fehlerakzeptierung

Über den Index **L_DRCERROROK** ist der Zugriff auf die DRC-Fehlerakzeptanzeinstellungen des aktuell geladenen Layoutelements möglich. Die Strukturdefinition von **L_DRCERROROK** lautet:

```
index L_DRCERROROK {            // Layout DRC-Fehlerakzeptierung Index
      string IDSTR;             // DRC-Fehler-Id-String
      };
```

L_DRILL - Layout Bohrung

Über den Index **L_DRILL** ist der Zugriff auf die Bohrdaten des aktuell geladenen Padstack-Bibliothekselements möglich. Die Strukturdefinition von **L_DRILL** lautet:

```
index L_DRILL {                 // Bohrung Index
      double X;                 // Bohrung X-Koordinate (STD2)
      double Y;                 // Bohrung Y-Koordinate (STD2)
      double RAD;               // Bohrung Radius (STD2)
      int CLASS;                // Bohrung Klasse (LAY5)
      };
```

Bei der Verwendung des Index **L_DRILL** in der Drillscanfunktion von **lay_scanfelem**, **lay_scanall** bzw. **lay_scanpool** ist zu beachten, dass in dem Index die Platzierungsdaten der Bohrung auf dem Padstack eingetragen sind. Die Koordinaten auf dem Layout bzw. Bauteil werden durch die übergebenen transformierten Koordinaten angegeben.

L_FIGURE - Layout Figurenelement

Über den Index **L_FIGURE** ist der Zugriff auf die platzierten Figurenelemente (Polygone, Leiterbahnen, Makroreferenzen, Texte, Bohrungen) des aktuell geladenen Elements möglich. Die Strukturdefinition von **L_FIGURE** lautet:

```
index L_FIGURE {                // Figurenelement Index
      int TYP;                  // Element Typ (LAY6)
      string NAME;              // Element Name
      double SIZE;              // Element Größe (STD2)
      double X;                 // Element X-Koordinate (STD2)
      double Y;                 // Element Y-Koordinate (STD2)
      double ANGLE;             // Element Drehwinkel (STD3)
      int MIRROR;               // Element Spiegelung (STD14)
      int LAYER;                // Element Lage/Klasse (LAY1 | LAY5)
      int GROUP;                // Element Gruppenflag (STD13)
      int FIXED;                // Element Fixiert Flag (STD11 | STD12)
      int TREE;                 // Element Netznummer
      index L_POOL POOL;        // Verweis auf Poolelement
      index L_POLY POLY;        // Verweis auf Polygonelement
      index L_LINE LINE;        // Verweis auf Leiterbahnelement
      index L_NREF NREF;        // Verweis auf Makroreferenz (benannt)
      index L_UREF UREF;        // Verweis auf Makroreferenz (unbenannt)
      index L_TEXT TEXT;        // Verweis auf Textelement
      index L_DRILL DRILL;      // Verweis auf Bohrelement
      };
```

Die Indexvariable **NAME** gibt für benannte Makroreferenzen den Namen und für Texte den Textstring an. Bei Elementen mit **TYP 7** (intern) gibt **NAME** den Namen des Padstackmakros zurück, falls es sich bei dem internen Element um eine Standardviadefinition handelt. Die Indexvariable **LAYER** gibt die Lagennummer des Figurenelements bzw. bei Bohrungen die Bohrklasse an. Die Indexvariable **POOL** ermöglicht den Zugriff auf das Bibliotheks- bzw. Poolelement, aus dem das Figurenelement aufgebaut ist. Einzelne Daten eines Figurenelements können mit den Funktionen **ged_elem*chg** modifiziert werden. Der komplette Datensatz eines Figurenelements kann mit der Funktion **lay_scanfelem** über alle Hierarchieebenen abgearbeitet werden.

L_LEVEL - Layout Signalpotential

Über den Index **L_LEVEL** ist der Zugriff auf die Connectivity-Level, d.h. auf die Netzlisten- bzw. Signalpotentiale des aktuell geladenen Layouts möglich. Die Strukturdefinition von **L_LEVEL** lautet:

```
index L_LEVEL {                 // Connectivity-Level Index
      int LEVVAL;               // Level Wert (LAY7)
      };
```

L_LINE - Layout Leiterbahn

Über den Index **L_LINE** ist der Zugriff auf die Leiterbahndaten des aktuell geladenen Layouts bzw. Bauteils möglich. Die Strukturdefinition von **L_LINE** lautet:

```
index L_LINE {                  // Leiterbahn Index
      double WIDTH;             // Leiterbahn Breite (STD2)
      int LAYER;                // Leiterbahn Lage (LAY1)
      int TREE;                 // Leiterbahn Netznummer
      int PN;                   // Anzahl Polygonpunkte
      index L_POINT;            // Liste der Polygonpunkte
      };
```

L_MACRO - Layout Bibliothekselement

Über den Index **L_MACRO** ist der Zugriff auf die Makros, d.h. auf die Bibliothekselemente (Bauteile, Padstacks, Pads) des aktuell geladenen Elements möglich. Die Strukturdefinition von **L_MACRO** lautet:

```
index L_MACRO {               // Makrodefinition Index
      string NAME;            // Makro Name
      double MLX;             // Linke Makrobegrenzung (STD2)
      double MLY;             // Untere Makrobegrenzung (STD2)
      double MUX;             // Rechte Makrobegrenzung (STD2)
      double MUY;             // Obere Makrobegrenzung (STD2)
      double MNX;             // Nullpunkt X-Koordinate (STD2)
      double MNY;             // Nullpunkt Y-Koordinate (STD2)
      int CLASS;              // Makro Klasse (STD1)
      int COMP;               // Makro Status (STD16)
      };
```

L_NREF - Layout Makroreferenz (benannt)

Über den Index **L_NREF** ist der Zugriff auf die benannten Makroreferenzen, d.h. auf die auf dem aktuell geladenen Element namentlich platzierten Bibliothekselemente möglich. Auf Layoutebene sind dies die platzierten Bauteile; auf Bauteilebene sind dies die platzierten Pins. Die Strukturdefinition von **L_NREF** lautet:

```
index L_NREF {                // Benannte Referenz Index
      string NAME;            // Referenz Name
      double X;               // Referenz X-Koordinate (STD2)
      double Y;               // Referenz Y-Koordinate (STD2)
      double ANGLE;           // Referenz Drehwinkel (STD3)
      int LAYOFF;             // Referenz Lagenoffset (LAY1)
      int MIRROR;             // Referenz Spiegelung (STD14)
      index L_MACRO MACRO;    // Makro Index
      };
```

Über die Indexvariable `MACRO` ist der Zugriff auf das durch das **L_NREF**-Element referenzierte Bibliothekselement möglich.

L_POINT - Layout Polygonpunkt

Über den Index **L_POINT** ist der Zugriff auf einzelne Polygonpunkte eines Polygons möglich. Die Strukturdefinition von **L_POINT** lautet:

```
index L_POINT {               // Polygonpunkt Index
      double X;               // Polygonpunkt X-Koordinate (STD2)
      double Y;               // Polygonpunkt Y-Koordinate (STD2)
      int TYP;                // Polygonpunkt Typ (STD15)
      };
```

Der Index **L_POINT** ist nur als `of`-Index für die Liste der Polygonpunkte in **L_POLY** bzw. in **L_LINE** zu verwenden.

L_POLY - Layout Polygon

Über den Index **L_POLY** ist der Zugriff auf die im aktuell geladenen Element definierten Polygone (Kupferflächen, Sperrflächen, Umrandung, Potentialflächen, Dokumentarlinien, Dokumentarflächen, Füllbereiche, Schraffurflächen oder Split-Powerplane-Flächen) möglich. Die Strukturdefinition von **L_POLY** lautet:

```
index L_POLY {                // Polygon Index
      int LAYER;              // Polygon Lage (LAY1)
      int TREE;               // Polygon Netznummer
      int TYP;                // Polygon Typ (LAY4)
      int MVIS;               // Polygon Spiegelungsmodus (LAY3) bzw:
                              //   LAY3 + 4 = gestricheltes Polygon
                              //   LAY3 + 8 = gepunktetes Polygon
      double WIDTH;           // Polygon Linienbreite (STD2)
      double DASHLEN;         // Polygon Strichelungslänge (STD2)
      double DASHSPC;         // Polygon Strichelungsrelativabstand
      int PN;                 // Anzahl Polygonpunkte
      index L_POINT;          // Liste der Polygonpunkte
      };
```

Der Zugriff auf die zum Polygon gehörenden Polygonpunkte erfolgt über eine entsprechende `forall-of`-Schleife. Die Netznummer ist nur für Potentialflächen, Füllbereiche, Schraffurflächen und Split-Powerplane-Flächen von Bedeutung. Zur Bestimmung der Netzzugehörigkeit passiver Kupferflächen sind die Funktionen **lay_scanfelem** bzw. **lay_scanall** zu verwenden.

L_POOL - Layout Poolelement

Über den Index **L_POOL** ist der Zugriff auf die aktuell geladenen Poolelemente möglich. Die Strukturdefinition von **L_POOL** lautet:

```
index L_POOL {                // Poolelement Index
      int TYP;                // Poolelement Typ (LAY8)
      int REFCNT;             // Poolelement Referenzierungsanzahl
      int LAYER;              // Poolelement Lage (LAY1)
      index L_POOL NXT;       // Verweis auf nächstes Poolelement
      index L_POOL REF;       // Verweis auf Referenz-Poolelement
      index L_POLY POLY;      // Verweis auf Polygonelement
      index L_LINE LINE;      // Verweis auf Leiterbahnelement
      index L_NREF NREF;      // Verweis auf Makroreferenz (benannt)
      index L_UREF UREF;      // Verweis auf Makroreferenz (unbenannt)
      index L_TEXT TEXT;      // Verweis auf Textelement
      index L_DRILL DRILL;    // Verweis auf Bohrelement
      index L_DRCERROR DRCERR;// Verweis auf DRC-Fehlerelement
      index L_MACRO MACRO;    // Verweis auf Bibliothekselement
      };
```

Der Index **L_POOL** wird zur Abarbeitung von Bibliotheksdefinitionen mit Hilfe der Systemfunktion **lay_scanpool** benötigt. Die Indexvariable `REFCNT` gibt an, wie oft das entsprechende Poolelement innerhalb des aktuell geladenen Elements referenziert wird. Die Indexvariablen `NXT` und `REF` dienen der schnellen Abarbeitung der aktuell referenzierten Poolelemente.

L_POWLAYER - Layout Versorgungslage

Über den Index **L_POWLAYER** ist der Zugriff auf die Definition der Versorgungslagen des aktuell geladenen Layouts möglich. Die Strukturdefinition von **L_POWLAYER** lautet:

```
index L_POWLAYER {            // Versorgungslage Index
      index L_CNET CNET;      // Verweis auf Netz
      index L_LEVEL LEVEL;    // Verweis auf Level
      int LAYER;              // Versorgungslagencode (LAY1)
      };
```

Die Indexvariable `CNET` ermöglicht den Zugriff auf das Netz, für welches die Versorgungslage definiert ist. Mit Hilfe der Indexvariablen `LEVEL` kann das Signalpotential der betreffenden Versorgungslage ermittelt werden.

L_TEXT - Layout Text

Über den Index **L_TEXT** ist der Zugriff auf die im aktuell geladenen Element definierten Texte möglich. Die Strukturdefinition von **L_TEXT** lautet:

```
index L_TEXT {                 // Text Index
      string STR;              // Text String
      double X;                // Text X-Koordinate (STD2)
      double Y;                // Text Y-Koordinate (STD2)
      double ANGLE;            // Text Drehwinkel (STD3)
      double SIZE;             // Text Größe (STD2)
      double WIDTH;            // Text Linienbreite (STD2)
      int LAYER;               // Text Lage (LAY1)
      int MIRROR;              // Text Spiegelung (STD14)
      int MODE;                // Text Modus (LAY2)
      };
```

L_UREF - Layout Makroreferenz (unbenannt)

Über den Index **L_UREF** ist der Zugriff auf die unbenannten Makroreferenzen, d.h. auf die auf dem aktuell geladenen Element namenlos platzierten Bibliothekselemente möglich. Auf Layout- bzw. Bauteilebene sind dies die platzierten Vias; auf Padstackebene sind dies die platzierten Pads. Die Strukturdefinition von **L_UREF** lautet:

```
index L_UREF {                 // Namenlose Referenz Index
      int TREE;                // Referenz Netznummer
      double X;                // Referenz X-Koordinate (STD2)
      double Y;                // Referenz Y-Koordinate (STD2)
      double ANGLE;            // Referenz Drehwinkel (STD3)
      int LAYOFF;              // Referenz Lagenoffset (LAY1)
      int MIRROR;              // Referenz Spiegelung (STD14)
      index L_MACRO MACRO;     // Verweis auf Makro
      };
```

Über die Indexvariable `MACRO` ist der Zugriff auf das durch das **L_UREF**-Element referenzierte Bibliothekselement möglich. Die Indexvariable `LAYOFF` ist nur für die Referenzierung von Pads auf Padstacks von Bedeutung.

B.5 CAM-View Index-Beschreibung (CV)

Dieser Abschnitt beschreibt die in der **Bartels User Language** definierten Index-Variablen-Typen für den **CAM-View** Datenzugriff (CV).

CV_DATASET - CAM-View Datensatz

Der Index **CV_DATASET** gestattet den Zugriff auf die aktuell geladenen **CAM-View**-Datensätze. Die Strukturdefinition von **CV_DATASET** lautet:

```
index CV_DATASET {            // Datensatz
      int IDX;                // Datensatz Index
      int TYP;                // Datensatz Typ
      int LLAYER;             // Datensatz Linienlage (LAY1)
      int FLAYER;             // Datensatz Blitzstrukturlage (LAY1)
      double XOFF;            // Datensatz X-Offset (STD2)
      double YOFF;            // Datensatz Y-Offset (STD2)
      int MIRROR;             // Datensatz Spiegelung (STD14)
      string NAME;            // Datensatz Dateiname
      };
```

B.6 IC Design Index-Beschreibung (ICD)

Dieser Abschnitt beschreibt die in der **Bartels User Language** definierten Index-Variablen-Typen für den **IC-Design** Datenzugriff (ICD).

I_ATTRIBUTE - IC-Design Netzlisteneintrag Attribut

Über den Index **I_ATTRIBUTE** ist der Zugriff auf die in der Netzliste eingetragenen Bauteil- bzw. Netzattribute des aktuell geladenen IC-Layouts möglich. Die Strukturdefinition von **I_ATTRIBUTE** lautet:

```
index I_ATTRIBUTE {            // Attribut Index
      string NAME;             // Attribut Name
      string VALUE;            // Attribut Wert
      };
```

I_CNET - IC-Design Netzliste

Über den Index **I_CNET** ist der Zugriff auf die in der Netzliste eingetragenen Netze des aktuell geladenen IC-Layouts möglich. Die Strukturdefinition von **I_CNET** lautet:

```
index I_CNET {                 // Physikalische Netzliste Index
      string NAME;             // Netz Name
      int NUMBER;              // Netz Nummer
      int PRIOR;               // Netz Routingpriorität
      double RDIST;            // Netz Mindestabstand (STD2)
      int PINN;                // Anzahl Pins
      index I_CPIN;            // Liste der Pins
      index I_ATTRIBUTE;       // Liste der Attribute
      index I_POOL UNRPOOL;    // Verweis auf Unroutes Poolelement
      };
```

Die Netznummer wird allgemein zur Identifizierung eines Netzes herangezogen. Mit der Funktion **icd_gettreeidx** kann ausgehend von der angegebenen Netznummer der entsprechende **I_CNET** Index gefunden werden. Der Mindestabstand gibt die Distanz an, die Leiterbahnen dieses Netzes zu nicht dem entsprechenden Netz zugehörenden Kupferstrukturen mindestens einhalten müssen. Der Zugriff auf die zum Netz gehörenden Pins bzw. Attribute erfolgt über eine entsprechende `forall-of`-Schleife. Die Indexvariable `UNRPOOL` ermöglicht den Zugriff auf die noch nicht verlegten Verbindungen (Unroutes, Airlines) des entsprechenden Netzes; die Abarbeitung dieser Verbindungen kann mit Hilfe der Systemfunktion **icd_scanpool** erfolgen.

I_CPART - IC-Design Netzlisteneintrag Bauteil

Über den Index **I_CPART** ist der Zugriff auf die in der Netzliste eingetragenen Bauteile des aktuell geladenen IC-Layouts möglich. Die Strukturdefinition von **I_CPART** lautet:

```
index I_CPART {                // Netzliste Bauteilindex
      string NAME;             // Bauteil Name
      string PLNAME;           // Physikalisches Bibliotheksteil
      int USED;                // Bauteilplatzierungscode:
                               //    0 = Bauteil nicht platziert
                               //    1 = Bauteil platziert
      int PEQUC;               // Bauteil Äquivalenz-Code
      int PINN;                // Anzahl Pins
      int FPINN;               // Anzahl nicht angeschlossene Pins
      index I_MACRO MACRO;     // Verweis auf Makro
      index I_CPIN;            // Liste der Pins
      index I_ATTRIBUTE;       // Liste der Attribute
      };
```

Auf die Pinliste und die Attributwerte kann mit Hilfe von entsprechenden `forall-of` Schleifen zugegriffen werden. Bauteile mit identischen Äquivalenz-Codes dürfen auf ihren Einbauplätzen im Zuge einer Platzierungsoptimierung vertauscht werden (Component Swap).

I_CPIN - IC-Design Netzlisteneintrag Bauteilpin

Über den Index **I_CPIN** ist der Zugriff auf die in der Netzliste eingetragenen Bauteilpins des aktuell geladenen IC-Layouts möglich. Die Strukturdefinition von **I_CPIN** lautet:

```
index I_CPIN {                  // Netzliste Pin Index
      string NAME;              // Pin Name
      double RWIDTH;            // Pin Routingbreite (STD2)
      int TREE;                 // Pin Netznummer
      int GATE;                 // Pin Gatternummer
      int GEQUC;                // Pin Gatter-Äquivalenz-Code
      int GEQUP;                // Pin Äquivalenz-Code
      int GGRPC;                // Pin Gatter-Gruppennummer
      int GPNUM;                // Pin Gatter-Relativnummer
      index I_CNET CNET;        // Verweis auf Netz
      index I_CPART CPART;      // Verweis auf Bauteil
      };
```

Die Pin Routingbreite gibt die Breite an, mit der von dem Pin zum nächsten Verbindungspunkt geroutet werden soll. Über die Indexvariablen **CNET** bzw. **CPART** ist der Rückwärtsverweis auf das Netz bzw. das Bauteil, an dem der Pin definiert ist, möglich. Über die Indexvariablen **GATE**, **GEQUC**, **GEQUP**, **GGRPC** und **GPNUM** kann die Zulässigkeit der Gatter- und Pinvertauschbarkeit (Pin/Gate Swap) ermittelt werden.

I_FIGURE - IC Design Figurenelement

Über den Index **I_FIGURE** ist der Zugriff auf die platzierten Figurenelemente (Polygone, Leiterbahnen, Makroreferenzen, Texte) des aktuell geladenen Elements möglich. Die Strukturdefinition von **I_FIGURE** lautet:

```
index I_FIGURE {                // Figurenelement Index
      int TYP;                  // Element Typ (ICD5)
      string NAME;              // Element Name
      double SIZE;              // Element Größe (STD2)
      double X;                 // Element X-Koordinate (STD2)
      double Y;                 // Element Y-Koordinate (STD2)
      double ANGLE;             // Element Drehwinkel (STD3)
      int MIRROR;               // Element Spiegelung (STD14)
      int LAYER;                // Element Lage (ICD1)
      int GROUP;                // Element Gruppenflag (STD13)
      int FIXED;                // Element Fixiert Flag (STD11)
      int TREE;                 // Element Netznummer
      int RULEOBJID;            // Element Regelsystemobjekt-Id
      index I_POOL POOL;        // Verweis auf Poolelement
      index I_POLY POLY;        // Verweis auf Polygonelement
      index I_LINE LINE;        // Verweis auf Leiterbahnelement
      index I_NREF NREF;        // Verweis auf Makroreferenz (benannt)
      index I_UREF UREF;        // Verweis auf Makroreferenz (unbenannt)
      index I_TEXT TEXT;        // Verweis auf Textelement
      };
```

Die Indexvariable **NAME** gibt für benannte Makroreferenzen den Namen und für Texte den Textstring an. Die Indexvariable **LAYER** gibt die Lagennummer des Figurenelements an. Die Indexvariable **POOL** ermöglicht den Zugriff auf das Bibliotheks- bzw. Poolelement, aus dem das Figurenelement aufgebaut ist. Einzelne Daten eines Figurenelements können mit den Funktionen **ced_elem*chg** modifiziert werden. Der komplette Datensatz eines Figurenelements kann mit der Funktion **icd_scanfelem** über alle Hierarchieebenen abgearbeitet werden.

I_LEVEL - IC Design Signalpotential

Über den Index **I_LEVEL** ist der Zugriff auf die Connectivity-Level, d.h. auf die Netzlisten- bzw. Signalpotentiale des aktuell geladenen IC-Layouts möglich. Die Strukturdefinition von **I_LEVEL** lautet:

```
index I_LEVEL {                 // Connectivity-Level Index
      int LEVVAL;               // Level Wert (ICD6)
      };
```

I_LINE - IC Design Leiterbahn

Über den Index **I_LINE** ist der Zugriff auf die Leiterbahndaten des aktuell geladenen IC-Layouts möglich. Die Strukturdefinition von **I_LINE** lautet:

```
index I_LINE {                // Leiterbahn Index
      double WIDTH;           // Leiterbahn Breite (STD2)
      int LAYER;              // Leiterbahn Lage (ICD1)
      int TREE;               // Leiterbahn Netznummer
      int PN;                 // Anzahl Polygonpunkte
      index I_POINT;          // Liste der Polygonpunkte
      };
```

I_MACRO - IC Design Bibliothekselement

Über den Index **I_MACRO** ist der Zugriff auf die Makros, d.h. auf die Bibliothekselemente (Bauteile, Padstacks, Pads) des aktuell geladenen Elements möglich. Die Strukturdefinition von **I_MACRO** lautet:

```
index I_MACRO {               // Makrodefinition Index
      string NAME;            // Makro Name
      double MLX;             // Linke Makrobegrenzung (STD2)
      double MLY;             // Untere Makrobegrenzung (STD2)
      double MUX;             // Rechte Makrobegrenzung (STD2)
      double MUY;             // Obere Makrobegrenzung (STD2)
      double MNX;             // Nullpunkt X-Koordinate (STD2)
      double MNY;             // Nullpunkt Y-Koordinate (STD2)
      int CLASS;              // Makro Klasse (STD1)
      int COMP;               // Makro Status (STD16)
      };
```

I_NREF - IC Design Makroreferenz (benannt)

Über den Index **I_NREF** ist der Zugriff auf die benannten Makroreferenzen, d.h. auf die auf dem aktuell geladenen Element namentlich platzierten Bibliothekselemente möglich. Auf IC-Layoutebene sind dies die platzierten Zellen; auf Zellenebene sind dies die platzierten Pins. Die Strukturdefinition von **I_NREF** lautet:

```
index I_NREF {                // Benannte Referenz Index
      string NAME;            // Referenz Name
      double X;               // Referenz X-Koordinate (STD2)
      double Y;               // Referenz Y-Koordinate (STD2)
      double ANGLE;           // Referenz Drehwinkel (STD3)
      double SCALE;           // Referenz Skalierungsfaktor
      int MIRROR;             // Referenz Spiegelung (STD14)
      index I_MACRO MACRO;    // Makro Index
      };
```

Über die Indexvariable `MACRO` ist der Zugriff auf das durch das **I_NREF**-Element referenzierte Bibliothekselement möglich.

I_POINT - IC Design Polygonpunkt

Über den Index **I_POINT** ist der Zugriff auf einzelne Polygonpunkte eines Polygons möglich. Die Strukturdefinition von **I_POINT** lautet:

```
index I_POINT {               // Polygonpunkt Index
      double X;               // Polygonpunkt X-Koordinate (STD2)
      double Y;               // Polygonpunkt Y-Koordinate (STD2)
      int TYP;                // Polygonpunkt Typ (STD15)
      };
```

Der Index **I_POINT** ist nur als `of`-Index für die Liste der Polygonpunkte in **I_POLY** bzw. in **I_LINE** zu verwenden.

I_POLY - IC Design Polygon

Über den Index **I_POLY** ist der Zugriff auf die im aktuell geladenen Element definierten Polygone (aktive Flächen, Sperrflächen oder Dokumentarlinien) möglich. Die Strukturdefinition von **I_POLY** lautet:

```
index I_POLY {                // Polygon Index
      int LAYER;              // Polygon Lage (ICD1)
      int TREE;               // Polygon Netznummer
      int TYP;                // Polygon Typ (ICD4)
      int MVIS;               // Polygon Spiegelungsmodus (ICD3)
      int PN;                 // Anzahl Polygonpunkte
      index I_POINT;          // Liste der Polygonpunkte
      };
```

Der Zugriff auf die zum Polygon gehörenden Polygonpunkte erfolgt über eine entsprechende **forall-of**-Schleife. Die Netznummer ist nur für aktive (leitende) Flächen von Bedeutung. Zur Bestimmung der Netzzugehörigkeit aktiver Flächen sind die Funktionen **icd_scanfelem** bzw. **icd_scanall** zu verwenden.

I_POOL - IC Design Poolelement

Über den Index **I_POOL** ist der Zugriff auf die aktuell geladenen Poolelemente möglich. Die Strukturdefinition von **I_POOL** lautet:

```
index I_POOL {                // Poolelement Index
      int TYP;                // Poolelement Typ (ICD7)
      int REFCNT;             // Poolelement Referenzierungsanzahl
      int LAYER;              // Poolelement Lage (ICD1)
      index I_POOL NXT;       // Verweis auf nächstes Poolelement
      index I_POOL REF;       // Verweis auf Referenz-Poolelement
      index I_POLY POLY;      // Verweis auf Polygonelement
      index I_LINE LINE;      // Verweis auf Leiterbahnelement
      index I_NREF NREF;      // Verweis auf Makroreferenz (benannt)
      index I_UREF UREF;      // Verweis auf Makroreferenz (unbenannt)
      index I_TEXT TEXT;      // Verweis auf Textelement
      index I_MACRO MACRO;    // Verweis auf Bibliothekselement
      };
```

Der Index **I_POOL** wird benötigt zur Abarbeitung von Bibliotheksdefinitionen mit Hilfe der Systemfunktion **icd_scanpool**. Die Indexvariable **REFCNT** gibt an, wie oft das entsprechende Poolelement innerhalb des aktuell geladenen Elements referenziert wird. Die Indexvariablen **NXT** und **REF** dienen der schnellen Abarbeitung der aktuell referenzierten Poolelemente.

I_TEXT - IC Design Text

Über den Index **I_TEXT** ist der Zugriff auf die im aktuell geladenen Element definierten Texte möglich. Die Strukturdefinition von **I_TEXT** lautet:

```
index I_TEXT {                // Text Index
      string STR;             // Text String
      double X;               // Text X-Koordinate (STD2)
      double Y;               // Text Y-Koordinate (STD2)
      double ANGLE;           // Text Drehwinkel (STD3)
      double SIZE;            // Text Größe (STD2)
      int LAYER;              // Text Lage (ICD1)
      int MIRROR;             // Text Spiegelung (STD14)
      int MODE;               // Text Modus (ICD2)
      };
```

I_UREF - IC Design Makroreferenz (unbenannt)

Über den Index **I_UREF** ist der Zugriff auf die unbenannten Makroreferenzen, d.h. auf die auf dem aktuell geladenen Element namenlos platzierten Bibliothekselemente möglich. Auf IC-Layoutebene sind dies die platzierten Vias. Die Strukturdefinition von **I_UREF** lautet:

```
index I_UREF {              // Namenlose Referenz Index
      int TREE;             // Referenz Netznummer
      double X;             // Referenz X-Koordinate (STD2)
      double Y;             // Referenz Y-Koordinate (STD2)
      double ANGLE;         // Referenz Drehwinkel (STD3)
      double SCALE;         // Referenz Skalierungsfaktor
      int MIRROR;           // Referenz Spiegelung (STD14)
      index I_MACRO MACRO;  // Verweis auf Makro
      };
```

Über die Indexvariable **MACRO** ist der Zugriff auf das durch das **I_UREF**-Element referenzierte Bibliothekselement möglich.

Anhang C Systemfunktionen

Dieser Anhang beschreibt die in der **Bartels User Language** eingebundenen Systemfunktionen. Die Funktionen sind dabei gegliedert nach den jeweiligen Aufruftypen in Referenzlisten aufgeführt und anschließend in Form eines Nachschlagewerks in alphabetischer Reihenfolge beschrieben.

Inhalt

C.1 Funktionsübersicht

Jede in der **Bartels User Language** definierte Systemfunktion ist genau einem der Aufruftypen STD, CAP, LAY, SCM, GED, AR, CAM, CV, ICD oder CED zugeordnet. Dieser Abschnitt enthält eine Übersicht über die zum jeweiligen Aufruftyp definierten Systemfunktionen.

Konventionen zur Funktionsbeschreibung

Für in den nachfolgenden Kapiteln dieses Anhangs beschriebene Funktion ist der zu dieser Funktion definierte Aufruftyp sowie eine formale Deklaration der Funktion und ihrer Parameter angegeben. Mit dem Datentyp der Systemfunktion ist auch der jeweilige Datentyp des zugehörigen Rückgabewertes definiert; ist der Datentyp einer Funktion `void`, dann gibt diese Funktion keinen Wert an den Aufrufer zurück. Wo nötig wird die Arbeitsweise einer Funktion durch nähere Angaben erläutert bzw. durch Beispiele veranschaulicht.

Bei der Deklaration der Parameter wird wenn nötig ein Wertebereich angegeben. Die Angabe erfolgt dabei mit unterer und oberer Grenze des Parameterbereichs. Für die untere Grenze sind folgende Angaben möglich:

`[ L`	Wert >= untere Grenze L
`] L`	Wert > untere Grenze L
`]`	keine Beschränkung nach unten

Für die obere Grenze sind entsprechend folgende Angaben möglich:

`U ]`	Wert <= obere Grenze U
`U [`	Wert < obere Grenze U
`[`	keine Beschränkung nach oben

Die Wertebereichsgrenzen werden durch `,` getrennt. Die Parameterdeklaration

```
double ]0.0,[;
```

gibt z.B. an, dass der Parameter vom Datentyp `double` und größer als 0.0 sein muss. Dem **User Language Compiler** sind die so definierten Parameterwertebereiche bekannt; er gibt ggf. eine Fehlermeldung aus, wenn er erkennt, dass ein Parameterwert außerhalb des gültigen Wertebereichs liegt.

Ist einem Parameter das Zeichen `&` vorangestellt, dann bedeutet dies, dass der Wert dieses Parameters durch die Systemfunktion gesetzt bzw. verändert wird. Dem **User Language Compiler** ist diese Parametereigenschaft bekannt; er gibt ggf. eine Warnmeldung aus, wenn er erkennt, dass ein konstanter Wert oder ein Berechnungsergebnis an einen solchen Parameter übergeben wird.

Ist einem Parameter das Zeichen `*` vorangestellt, dann handelt es sich bei diesem Parameter um die Referenz auf eine Anwenderfunktion; in diesem Fall wird zusätzlich die vorgeschriebene Deklaration dieser Anwenderfunktion mit aufgeführt. Die Systemfunktion aktiviert von sich aus die entsprechende Anwenderfunktion, sofern diese im **User Language**-Programm definiert ist. Unter Umständen ist die Referenzierung von Anwenderfunktionen optional; wenn in diesem Fall eine entsprechende Anwenderfunktion nicht aktiviert werden soll, dann ist für den entsprechenden Parameter das Schlüsselwort `NULL` einzutragen. Bei der Deklaration referenzierter Anwenderfunktionen sollte mit größter Sorgfalt vorgegangen werden, da erst zur Laufzeit (also durch den **User Language Interpreter**) festgestellt werden kann, ob die Konventionen hinsichtlich des Datentyps und der Parameterdeklarationen für die Anwenderfunktion eingehalten wurden (ist dies nicht der Fall, dann kommt es zu einem Laufzeitfehler); auch das Setzen des Rückgabewertes der referenzierten Anwenderfunktion sollte unbedingt entsprechend den vorgegebenen Konventionen erfolgen, da sich sonst u.U. fatale Nebeneffekte ergeben, die man auf Anhieb gar nicht erkennen (geschweige denn abfangen) kann.

Einen weiteren Spezialfall für die Parameter von Systemfunktionen stellt schließlich der Datentyp `void` dar, der angibt, dass der betreffende Parameter von einem beliebigen Datentyp sein kann. Die Spezifikation des Parametertyps `[]` schließlich gibt an, dass die Funktion an dieser Stelle optional eine beliebige Anzahl von Parametern vom Datentyp `void` erwartet.

C.1.1 Standard Systemfunktionen (STD)

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp STD zugeordnet, d.h. diese Funktionen können in allen definierten Interpreterumgebungen aufgerufen werden:

abs	Absolutwert eines Ganzzahlwertes
acos	Arcuscosinus berechnen
angclass	Winkelklassifizierung
arylength	Arraylänge bestimmen
asin	Arcussinus berechnen
askcoord	Benutzereingabe X-/Y-Koordinatenwerte
askdbl	Benutzereingabe Gleitkommazahlwert
askdist	Benutzereingabe Distanzwert
askint	Benutzereingabe Ganzzahlwert
askstr	Benutzereingabe Zeichenkette
atan	Arcustangens berechnen
atan2	Arcustangens eines punktbestimmten Winkels
atof	Umwandlung Zeichenkette in Gleitkommazahlwert
atoi	Umwandlung Zeichenkette in Ganzzahlwert
bae_askddbename	BAE DDB-Elementnamensabfrage
bae_askddbfname	BAE DDB-Dateinamensabfrage
bae_askdirname	BAE Dateiverzeichnisnamensabfrage
bae_askfilename	BAE Dateinamensabfrage
bae_askmenu	BAE Menüabfrage
bae_askname	BAE Dialog zur Namensauswahl aktivieren
bae_asksymname	BAE DDB-Bibliothekselementabfrage
bae_callmenu	BAE Menüfunktion aufrufen
bae_charsize	BAE Text-/Zeichengröße abfragen
bae_cleardistpoly	Internes BAE Distanzabfragepolygon löschen
bae_clearpoints	BAE Polygonpunktliste löschen
bae_clriactqueue	BAE Interaktionsvorgaben löschen
bae_crossarcarc	Schnittpunkt(e) zwischen Kreisbögen bestimmen
bae_crosslineline	Schnittpunkt zwischen Liniensegmenten mit Breite bestimmen
bae_crosslinepoly	Schnittpunkt zwischen Liniensegment mit Breite und Polygon bestimmen
bae_crosssegarc	Schnittpunkt zwischen Liniensegment und Kreisbogen bestimmen
bae_crosssegseg	Schnittpunkt zwischen Liniensegmenten/Linien bestimmen
bae_dashpolyline	Gestricheltes BAE Polygon vektorisieren

bae_deffuncprog	BAE Funktionstaste programmieren
bae_defkeyprog	BAE Standardtaste programmieren
bae_defmenu	BAE Standardmenüdefinition starten
bae_defmenuprog	BAE Menüfunktion programmieren
bae_defmenusel	BAE Menüabfrage Vorauswahl
bae_defmenutext	BAE Menütext definieren
bae_defselmenu	BAE Submenüdefinition starten
bae_dialaddcontrol	BAE Dialogelement definieren
bae_dialadvcontrol	Erweitertes BAE-Dialogelement setzen
bae_dialaskcall	BAE Dialog mit Listboxelement-Callbackfunktion aktivieren
bae_dialaskparams	BAE Dialog aktivieren
bae_dialbmpalloc	BAE Dialogbitmap erzeugen
bae_dialboxbufload	BAE Dialogboxdaten aus Buffer holen
bae_dialboxbufstore	BAE Dialogboxdaten in Buffer sichern
bae_dialboxperm	Eigenständigen BAE Dialog aktivieren
bae_dialclr	BAE Dialogelemente löschen
bae_dialgetdata	BAE Dialogelementdaten abfragen
bae_dialgettextlen	BAE Dialogtextlänge abfragen
bae_dialsetcurrent	Aktuelle BAE-Dialogbox setzen
bae_dialsetdata	BAE Dialogelementdaten setzen
bae_endmainmenu	BAE Hauptmenüdefinition beenden
bae_endmenu	BAE Menüdefinition beenden
bae_fontcharcnt	BAE Anzahl Zeichensatzzeichen
bae_fontname	BAE Zeichensatzname abfragen
bae_getactmenu	BAE aktives Menü abfragen
bae_getanglelock	BAE Winkelfreigabeflag abfragen
bae_getbackgrid	BAE Hintergrundraster abfragen
bae_getcasstime	Zeitpunkt des letzten Projektnetzlistenupdates durch Packager/Backannotation ermitteln
bae_getclassbitfield	BAE DDB-Klasse Bearbeitungsschlüssel abfragen
bae_getcmdbuf	BAE Kommandohistorie abfragen
bae_getcolor	BAE Farbwert abfragen
bae_getcoorddisp	BAE Koordinatenanzeige abfragen
bae_getdblpar	BAE Doubleparameter abfragen
bae_getfuncprog	BAE Funktionstastenprogrammierung abfragen
bae_getgridlock	BAE Rasterfreigabeflag abfragen
bae_getgridmode	BAE Rasterabhängigkeitsmodus abfragen

bae_getinpgrid	BAE Eingaberaster abfragen
bae_getintpar	BAE Integerparameter abfragen
bae_getinvcolor	BAE Farbinvertierungsmodus abfragen
bae_getkeyprog	BAE Standardtastenprogrammierung abfragen
bae_getmenubitfield	BAE Menüfunktion Bearbeitungsschlüssel abfragen
bae_getmenuitem	BAE Menüeintrag abfragen
bae_getmenuprog	BAE Menüprogrammierung abfragen
bae_getmenutext	BAE Menütext abfragen
bae_getmoduleid	BAE Modulbezeichnung abfragen
bae_getmsg	BAE HighEnd Message empfangen
bae_getpackdata	Daten des letzten Projekt-Packagerlaufs ermitteln
bae_getpacktime	Datum/Uhrzeit des letzten Projekt-Packagerlaufs ermitteln
bae_getpolyrange	Bereich der internen BAE-Polygonpunktliste abfragen
bae_getstrpar	BAE Stringparameter abfragen
bae_inittextscreen	BAE Textbildschirm initialisieren/löschen
bae_inpoint	BAE Eingabe Punkt mit Maus
bae_inpointmenu	BAE Eingabe Punkt mit Maus und Callbackfunktion für rechte Maustaste
bae_language	BAE Benutzeroberfläche Landessprache abfragen
bae_loadcoltab	BAE Farbtabelle laden
bae_loadelem	BAE Element laden
bae_loadfont	BAE Zeichensatz laden
bae_menuitemhelp	Onlinehilfe zu BAE-Menüelement anzeigen
bae_msgbox	BAE Info-Popup aktivieren
bae_msgboxverify	BAE Info-Popup mit Ja/Nein-Abfrage aktivieren
bae_msgboxverifyquit	BAE Info-Popup mit Ja/Nein/Abbruch-Abfrage aktivieren
bae_msgprogressrep	BAE-Fortschrittsanzeige aktivieren/aktualisieren
bae_msgprogressterm	BAE-Fortschrittsanzeige beenden
bae_mtpsize	BAE Popup Anzeigebereichsdimensionen abfragen
bae_nameadd	BAE Namensauswahlliste Element hinzufügen
bae_nameclr	BAE Namensauswahlliste löschen
bae_nameget	BAE Namensauswahlliste Element abfragen
bae_numstring	Numerische Zeichenkette erzeugen
bae_peekiact	BAE Interaktionsvorgaben abfrage
bae_plainmenutext	BAE Menütext konvertieren
bae_planddbclass	BAE Elementklasse abfragen
bae_planename	BAE Elementname abfragen

bae_planfname	BAE Dateiname abfragen
bae_plannotsaved	BAE Element ungesichert Flag abfragen
bae_plansename	BAE Zielelementname abfragen
bae_plansfname	BAE Zieldateiname abfragen
bae_planwslx	BAE Element linke Elementgrenze abfragen
bae_planwsly	BAE Element untere Elementgrenze abfragen
bae_planwsnx	BAE Element X-Bezugskoordinate abfragen
bae_planwsny	BAE Element Y-Bezugskoordinate abfragen
bae_planwsux	BAE Element rechte Elementgrenze abfragen
bae_planwsuy	BAE Element obere Elementgrenze abfragen
bae_popareachoice	BAE Popupmenü Selektionsbereich definieren
bae_popcliparea	BAE Popupmenü Clippingbereich definieren
bae_popclrtool	BAE Toolbar-Popupmenübereich löschen
bae_popcolbar	BAE Popupmenü Farbbalkenanzeige definieren
bae_popcolchoice	BAE Popupmenü Farbbalkenselektion definieren
bae_popdrawpoly	BAE Popupmenü Polygon-/Grafikanzeige
bae_popdrawtext	BAE Popupmenü Textanzeige
bae_popmouse	BAE Popup/Toolbar Mausposition abfragen
bae_poprestore	BAE Popupmenübereich reaktivieren
bae_popsetarea	BAE Popupmenübereich aktivieren/selektieren
bae_popshow	BAE Popupmenü aktivieren
bae_poptext	BAE Popupmenü Textanzeige definieren
bae_poptextchoice	BAE Popupmenü Textselektion definieren
bae_postprocess	BAE Postprozessorlauf
bae_progdir	BAE Programmverzeichnis ermitteln
bae_prtdialog	BAE Dialogtextausgabe in Statuszeile
bae_querydist	BAE Punkt-zu-Polygon Distanzabfrage
bae_readedittext	BAE Texteingabe/-anzeige
bae_readtext	BAE Texteingabe mit optionalem Popupmenü
bae_redefmainmenu	BAE Hauptmenüdefinition starten
bae_redefmenu	BAE Menüfunktion umprogrammieren
bae_resetmenuprog	BAE Menüprogrammierung zurücksetzen
bae_sendmsg	BAE HighEnd Message senden
bae_setanglelock	BAE Winkelfreigabeflag setzen
bae_setbackgrid	BAE Hintergrundraster setzen
bae_setclipboard	Textstring in BAE-Zwischenablage speichern

bae_setcolor	BAE Farbwert setzen
bae_setcoorddisp	BAE Koordinatenanzeige setzen
bae_setdblpar	BAE Doubleparameter setzen
bae_setgridlock	BAE Rasterfreigabeflag setzen
bae_setgridmode	BAE Rasterabhängigkeitsmodus setzen
bae_setinpgrid	BAE Eingaberaster setzen
bae_setintpar	BAE Integerparameter setzen
bae_setmoduleid	BAE Modulbezeichnung setzen
bae_setmousetext	BAE Mausklick-Eingabetext definieren
bae_setplanfname	BAE Projektdateiname setzen
bae_setpopdash	BAE Popup/Toolbar Parameter für gestrichelte Linien setzen
bae_setstrpar	BAE Stringparameter setzen
bae_settbsize	BAE Toolbarbereich definieren/anzeigen
bae_storecmdbuf	BAE Kommando in Kommandohistorie speichern
bae_storedistpoly	Internes BAE Distanzabfragepolygon speichern
bae_storeelem	BAE Element speichern
bae_storekeyiact	BAE Tasteneingabe vorgeben
bae_storemenuiact	BAE Menüwahl vorgeben
bae_storemouseiact	BAE Mauseingabe vorgeben
bae_storepoint	Punkt in BAE-Punktliste eintragen
bae_storetextiact	BAE Texteingabe vorgeben
bae_swconfig	BAE Softwarekonfiguration abfragen
bae_swversion	BAE Softwareversion abfragen
bae_tbsize	BAE Toolbardimensionen abfragen
bae_twsize	BAE Textarbeitsbereichsgröße abfragen
bae_wsmouse	BAE Arbeitsbereich Mausposition abfragen
bae_wswinlx	BAE Arbeitsbereich linke Grenze abfragen
bae_wswinly	BAE Arbeitsbereich untere Grenze abfragen
bae_wswinux	BAE Arbeitsbereich rechte Grenze abfragen
bae_wswinuy	BAE Arbeitsbereich obere Grenze abfragen
catext	Dateinamenserweiterung an Dateiname anhängen
catextadv	Dateinamenserweiterung optional an Dateiname anhängen
ceil	Gleitkommazahl aufrunden
clock	Verbrauchte CPU-Zeit ermitteln
con_clear	Interne Logische Netzliste löschen
con_compileloglib	Logische Bibliotheksdefinition kompilieren

con_deflogpart	Logische Bauteildefinition speichern
con_getddbpattrib	Bauteil-/Pinattribut in DDB-Datei abfragen
con_getlogpart	Logische Bauteildefinition abfragen
con_setddbpattrib	Bauteil-/Pinattribut in DDB-Datei setzen
con_storepart	Interne Logische Netzliste Bauteil speichern
con_storepin	Interne Logische Netzliste Pin speichern
con_write	Interne Logische Netzliste auf Datei ausgeben
convstring	Zeichenkette konvertieren
cos	Cosinus berechnen
cosh	Hyberbolischen Cosinus berechnen
cvtangle	Winkel in andere Einheit umwandeln
cvtlength	Länge in andere Einheit umwandeln
ddbcheck	DDB-Dateielement auf Verfügbarkeit prüfen
ddbclassid	DDB-Elementklasse Bezeichnung abfragen
ddbclassscan	DDB-Elementklasse abarbeiten
ddbcopyelem	DDB-Dateielement kopieren
ddbdelelem	DDB-Dateielement löschen
ddbelemrefcount	DDB-Dateielement Referenzanzahl abfragen
ddbelemrefentry	DDB-Dateielement Referenzeintrag abfragen
ddbgetelemcomment	DDB-Dateielement Kommentartext abfragen
ddbgetlaypartpin	DDB-Dateielement Layoutbauteilpindaten abfragen
ddbrenameelem	DDB-Dateielement umbenennen
ddbupdtime	DDB-Dateielement Änderungsdatum abfragen
ddbsetelemcomment	DDB-Dateielement Kommentartext setzen
dirscan	Dateiverzeichniseinträge abarbeiten
existddbelem	DDB-Dateielement Existenz prüfen
exit	Programm verlassen
exp	Exponentialfunktion
fabs	Absolutwert eines Gleitkommawertes
fclose	Datei schließen
fcloseall	Alle offenen Dateien schließen
feof	Prüfen ob Dateiende erreicht
fgetc	Zeichen aus Datei einlesen
fgets	Zeichenkette aus Datei einlesen
filemode	Dateimodus abfragen
filesize	Dateigröße abfragen

filetype	Dateityp abfragen
floor	Gleitkommawert abrunden
fmod	Gleitkommadivision Rest berechnen
fopen	Datei öffnen
fprintf	Formatierte Ausgabe auf Datei
fputc	Zeichen in Datei schreiben
fputs	Zeichenkette in Datei schreiben
frexp	Exponentialdarstellung ermitteln
fseterrmode	Dateifehler-Behandlungsmodus setzen
get_date	Systemdatum ermitteln
get_time	Systemuhrzeit ermitteln
getchr	Zeichen einlesen
getcwd	Pfadname des Arbeitsverzeichnisses abfragen
getenv	Umgebungsvariable abfragen
getextprog	Dateitypspezifische Applikation ermitteln
getstr	Zeichenkette einlesen
isalnum	Prüfen ob Zeichen alphanumerisch
isalpha	Prüfen ob Zeichen Buchstabe
iscntrl	Prüfen ob Zeichen Kontrollzeichen
isdigit	Prüfen ob Zeichen Ziffer
isgraph	Prüfen ob Zeichen sichtbares Zeichen
islower	Prüfen ob Zeichen Kleinbuchstabe
isprint	Prüfen ob Zeichen druckbares Zeichen
ispunct	Prüfen ob Zeichen Satzzeichen
isspace	Prüfen ob Zeichen Zwischenraumzeichen
isupper	Prüfen ob Zeichen Großbuchstabe
isxdigit	Prüfen ob Zeichen Hex-Ziffer
kbhit	Prüfen ob Taste betätigt
kbstate	Umschalt-/Steuerungs-Tastenstatus abfragen
launch	Betriebssystemkommando absetzen ohne Ausführung abzuwarten
ldexp	Gleitkommamultiplikation mit 2^n
localtime	Systemdatum und Systemzeit abfragen
log	Logarithmus zur Basis e
log10	Logarithmus zur Basis 10
mkdir	Dateiverzeichnis anlegen
modf	Gleitkommazahl Vor- und Nachkommastellen

namestrcmp	Namensvergleich
numstrcmp	Numerischer Zeichenkettenvergleich
perror	Fehlermeldung in Statuszeile ausgeben
pow	Potenzfunktion x^y
printf	Formatierte Ausgabe
programid	Programmname abfragen
putchr	Zeichen ausgeben
putenv	Umgebungsvariable setzen
puts	Zeichenkette mit Zeilenabschluss ausgeben
putstr	Zeichenkette ausgeben
quicksort	Indexliste sortieren
remove	Datei oder Verzeichnis löschen
rename	Datei umbenennen
rewind	Auf Dateianfang positionieren
rulecompile	Regeldefinition kompilieren
rulesource	Regeldefinitionsquellcode abfragen
scanddbenames	Inhalt Datenbank abfragen
scandirfnames	Inhalt Dateiverzeichnis abfragen
setprio	BAE Prozesspriorität setzen
sin	Sinus berechnen
sinh	Hyberbolischen Sinus berechnen
sprintf	Formatierte Ausgabe auf Zeichenkette
sqlcmd	SQL Kommando ausführen
sqlerr	SQL Fehlerstatus abfragen
sqlinit	SQL Datenbank initialisieren
sqrt	Quadratwurzel berechnen
strcmp	ASCII-Zeichenkettenvergleich
strcspn	Zeichenkette Länge Startmuster berechnen
strdelchr	Zeichenkette Zeichenmenge entfernen
strextract	Zeichenkette Sub-Zeichenkette extrahieren
strextractfilepath	Verzeichnisname aus Dateipfadname extrahieren
strgetconffilename	Konfigurationsdateiname mit optionaler Umgebungsvariable bestimmen
strgetvarfilename	Dateiname aus Umgebungsvariable ableiten
strgetpurefilename	Dateiname aus Dateipfadname extrahieren
strlen	Länge Zeichenkette ermitteln
strlistitemadd	String in Stringliste eintragen

strlistitemchk	String in Stringliste suchen
strlower	Zeichenkette in Kleinbuchstaben umwandeln
strmatch	Zeichenkette Musterabfrage
strnset	Zeichenkette mit n Zeichen füllen
strreverse	Zeichenkette Zeichenreihenfolge umdrehen
strscannext	Zeichen in Zeichenkette vorwärts suchen
strscanprior	Zeichen in Zeichenkette rückwärts suchen
strset	Zeichenkette mit Zeichen füllen
strspn	Zeichenkette Länge bis Endmuster ermitteln
strupper	Zeichenkette in Großbuchstaben umwandeln
syngetintpar	BNF/Scanner Integerparameter abfragen
synparsefile	BNF/Parser Datei einlesen
synparseincfile	BNF/Parser Includedatei einlesen
synparsestring	BNF/Parser Zeichenkette abarbeiten
synscaneoln	BNF/Scanner Zeilenendeerkennung setzen
synscanigncase	BNF/Scanner Schlüsselworterkennungsmodus setzen
synscanline	BNF/Scanner Eingabezeilennummer abfragen
synscanstring	BNF/Scanner Eingabestring abfragen
synsetintpar	BNF/Scanner Integerparameter setzen
system	Betriebssystemkommando absetzen und Ausführung abwarten
tan	Tangens berechnen
tanh	Hyberbolischen Tangens berechnen
tolower	Zeichen in Kleinbuchstaben umwandeln
toupper	Zeichen in Großbuchstaben umwandeln
uliptype	User Language Interpreterumgebung abfragen
ulipversion	User Language-Interpreterversion abfragen
ulproginfo	User Language-Programm Info abfragen
ulsystem	User Language-Programm aufrufen
ulsystem_exit	User Language-Programm nach Beendigung des aktuellen User Language-Programms aufrufen
vardelete	Globale User Language-Variable löschen
varget	Globale User Language-Variable abfragen
varset	Globale User Language-Variable setzen

C.1.2 Schematic Capture Systemfunktionen (CAP)

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp CAP zugeordnet, d.h. diese Funktionen können im **Schaltplaneditor** aufgerufen werden:

cap_blockname	Schaltplan Blockname abfragen
cap_blocktopflag	Schaltplan Blockhierarchieebene abfragen
cap_figboxtest	SCM-Elementüberschneidung Rechteck prüfen
cap_findblockname	SCM-Blockschaltbild mit angegebenem Blocknamen suchen
cap_findlayconpart	Bauteilindex aus Layoutnetzliste ermitteln
cap_findlayconpartpin	Bauteilpinindex aus Layoutnetzliste ermitteln
cap_findlaycontree	Netznamens-Netzindex aus Layoutnetzliste ermitteln
cap_getglobnetref	Globale Netznamensreferenz abfragen
cap_getlaytreeidx	Netznummer-Netzindex aus Layoutnetzliste ermitteln
cap_getpartattrib	SCM-Bauteilattributwert abfragen
cap_getrulecnt	SCM-Element Regelanzahl abfragen
cap_getrulename	SCM-Element Regelname abfragen
cap_getscbustapidx	Aktuell gescannten SCM-Busanschluß ermitteln
cap_getscclass	Aktuell gescannte SCM-Elementklasse ermitteln
cap_getscrefpidx	Aktuell gescanntes SCM-Bibliothekselement ermitteln
cap_getscstkcnt	Schaltplan Scanfunktion Stacktiefe abfragen
cap_gettagdata	Schaltplan Tagsymbol Zieldaten abfragen
cap_lastconseg	Zuletzt modifiziertes SCM-Verbindungssegment ermitteln
cap_lastfigelem	Zuletzt modifiziertes SCM-Element ermitteln
cap_layconload	Layoutnetzliste laden
cap_maccoords	Schaltplan Makrokoordinaten abfragen
cap_macload	SCM-Symbol in den Arbeitsspeicher laden
cap_macrelease	SCM-Symbol aus Arbeitsspeicher löschen
cap_mactaglink	Schaltplan Makro-Tagverweisdaten abfragen
cap_nrefsearch	Schaltplan Name auf Plan suchen
cap_partplan	Schaltplan Bauteilplanname abfragen
cap_pointpoolidx	Schaltplan Verbindungspunktpoolelement ermitteln
cap_ruleconatt	SCM-Regelsystem Fehlerstatus abfragen
cap_rulecondet	Regelzuweisung an SCM-Verbindungssegment
cap_ruleerr	Regelzuweisungen von SCM-Verbindungssegment lösen
cap_rulefigatt	Regelzuweisung an SCM-Figurenelement
cap_rulefigdet	Regelzuweisungen von SCM-Figurenelement lösen

cap_ruleplanatt	Regelzuweisung an aktuell geladenes SCM-Element
cap_ruleplandet	Regelzuweisungen von aktuell geladenem SCM-Element lösen
cap_rulequery	SCM-Element Regelabfrage durchführen
cap_scanall	Schaltplan Scan über alle Elemente
cap_scanfelem	Schaltplan Scan über Figurenelement
cap_scanpool	Schaltplan Scan über Poolelement
cap_vecttext	Schaltplan Text vektorisieren

C.1.3 Schematic Editor Systemfunktionen (SCM)

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp SCM zugeordnet, d.h. diese Funktionen können im **Schaltplaneditor** aufgerufen werden:

scm_askrefname	SCM Referenznamensabfrage
scm_asktreenam	SCM Netznamensabfrage
scm_attachtextpos	Textverschiebung an SCM-Element anfügen
scm_checkbustapplot	SCM-Busanschluß Plotstatus abfragen
scm_checkjunctplot	SCM-Verbindungspunktmarker Plotstatus abfragen
scm_chkattrname	SCM-Attributname validieren
scm_conseggrpchg	SCM Verbindungssegment Gruppenflag ändern
scm_deflibname	SCM Setup default Bibliothek
scm_defloglname	SCM Setup default Packager Bibliothek
scm_defsegbus	SCM Verbindungssegment Busdefinition
scm_delconseg	SCM Verbindungssegment löschen
scm_delelem	SCM Element löschen
scm_drawelem	SCM Elementanzeige aktualisieren
scm_elemangchg	SCM Elementwinkel ändern
scm_elemgrpchg	SCM Element Gruppenflag ändern
scm_elemmirrchg	SCM Elementspiegelung ändern
scm_elemposchg	SCM Elementposition ändern
scm_elemsizechg	SCM Elementgröße ändern
scm_findpartplc	Layoutbauteil Platzierungsstatus abfragen (BAE HighEnd)
scm_getdblpar	SCM Doubleparameter abfragen
scm_getgroupdata	SCM Gruppenplatzierungsdaten abfragen
scm_gethighlnet	SCM Netz Highlightmodus abfragen
scm_gethpglparam	SCM HP-GL-Plotparameter abfragen
scm_getinputdata	SCM Eingabedaten abfragen
scm_getintpar	SCM Integerparameter abfragen
scm_getstrpar	SCM Stringparameter abfragen
scm_highlnet	SCM Netz Highlightmodus setzen
scm_pickanyelem	Beliebiges SCM Element selektieren
scm_pickbustap	SCM Bustap selektieren
scm_pickconseg	SCM Verbindungssegment selektieren
scm_pickelem	SCM Element selektieren
scm_setdblpar	SCM Doubleparameter setzen

scm_setintpar	SCM Integerparameter setzen
scm_setpartattrib	SCM Bauteilattribut setzen
scm_setpickconseg	SCM Defaultverbindungspickelement setzen
scm_setpickelem	SCM Defaultpickelement setzen
scm_setstrpar	SCM Stringparameter setzen
scm_settagdata	SCM Tagsymbolpin Zieldaten setzen
scm_storecon	SCM Verbindung platzieren
scm_storelabel	SCM Label platzieren
scm_storepart	SCM Bauteil platzieren
scm_storepoly	SCM Polygon platzieren
scm_storetext	SCM Text platzieren

C.1.4 Layout Systemfunktionen (LAY)

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp LAY zugeordnet, d.h. diese Funktionen können im **Layouteditor**, im **Autorouter** und im **CAM-Prozessor** aufgerufen werden:

lay_defelemname	Layout Setup default Elementname
lay_deflibname	Layout Setup default Bibliothek
lay_defusrunit	Layout Setup default Benutzereinheitensystem
lay_doclayindex	Layout Dokumentarlagenanzeigeindex
lay_doclayname	Layout Setup Name Dokumentarlage
lay_doclayside	Layout Setup Seitenmodus Dokumentarlage
lay_doclaytext	Layout Setup Textmodus Dokumentarlage
lay_figboxtest	Layout Elementüberschneidung Rechteck prüfen
lay_findconpart	Layout Bauteil in Netzliste suchen
lay_findconpartpin	Layout Bauteilpin in Netzliste suchen
lay_findcontree	Layout Netz in Netzliste suchen
lay_getplanchkparam	Layout DRC Abstände abfragen
lay_getpowplanetree	Layout Netznummer in Versorgungslage abfragen
lay_getpowpolystat	Layout Versorgungslagenpolygonstatus abfragen
lay_getrulecnt	Layoutelement Regelanzahl abfragen
lay_getrulename	Layoutelement Regelname abfragen
lay_getscclass	Aktuell gescannte Layoutelementklasse ermitteln
lay_getscpartrpidx	Aktuell gescanntes Layoutbauteil ermitteln
lay_getscrefpidx	Aktuell gescanntes Layoutbibliothekselement ermitteln
lay_getscstkcnt	Layout Scanfunktion Stacktiefe abfragen
lay_getsctextdest	Zielpunkt des gescannten Layouttextes abfragen
lay_gettreeidx	Layout Netznummer in Netzliste suchen
lay_grpdisplay	Layout Setup Gruppenlage abfragen
lay_lastfigelem	Zuletzt modifiziertes Layoutelement ermitteln
lay_maccoords	Layout Makrokoordinaten abfragen
lay_macload	Layoutsymbol in den Arbeitsspeicher laden
lay_macrelease	Layoutsymbol aus Arbeitsspeicher löschen
lay_menulaylinecnt	Lagenmenüzeilenanzahl abfragen
lay_menulaylinelay	Lagennummer der angegebenen Lagenmenüzeile abfragen
lay_menulaylinename	Lagenname der angegebenen Lagenmenüzeile abfragen
lay_nrefsearch	Layout Name auf Plan suchen
lay_planmidlaycnt	Layout Innenlagenanzahl abfragen

lay_plantoplay	Layout oberste Lage abfragen
lay_pltmarklay	Layout Setup Passermarkenlage abfragen
lay_ruleerr	Layout-Regelsystem Fehlerstatus abfragen
lay_rulefigatt	Regelzuweisung an Layout-Figurenelement
lay_rulefigdet	Regelzuweisungen von Layout-Figurenelement lösen
lay_rulelaysatt	Regelzuweisung an Layoutlagenaufbau
lay_rulelaysdet	Regelzuweisungen von Layoutlagenaufbau lösen
lay_ruleplanatt	Regelzuweisung an aktuell geladenes Layoutelement
lay_ruleplandet	Regelzuweisungen von aktuell geladenem Layoutelement lösen
lay_rulequery	Layoutelement Regelabfrage durchführen
lay_scanall	Layout Scan über alle Elemente
lay_scanfelem	Layout Scan über Figurenelement
lay_scanpool	Layout Scan über Poolelement
lay_setfigcache	Layout-Cache für den schnellen Zugriff auf Figurenlistenelemente aufbauen
lay_setplanchkparam	Layout DRC Parameter setzen
lay_toplayname	Layout Setup Name oberste Lage abfragen
lay_vecttext	Layout Text vektorisieren

C.1.5 Layouteditor Systemfunktionen (GED)

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp GED zugeordnet, d.h. diese Funktionen können im **Layouteditor** aufgerufen werden:

ged_asklayer	GED Lagenauswahl
ged_askrefname	GED Referenznamensabfrage
ged_asktreeidx	GED Netzabfrage
ged_attachtextpos	Textverschiebung an Layoutelement anfügen
ged_delelem	GED Element löschen
ged_drawelem	GED Elementanzeige aktualisieren
ged_drcerrorhide	GED DRC-Fehlerakzeptanzmodus setzen/rücksetzen
ged_drcpath	GED Designregelprüfung für Leiterbahn-Testplatzierung
ged_drcpoly	GED Designregelprüfung für Polygon-Testplatzierung
ged_drcvia	GED Designregelprüfung für Via-Testplatzierung
ged_elemangchg	GED Elementwinkel ändern
ged_elemfixchg	GED Element fixiert-Flag ändern
ged_elemgrpchg	GED Element Gruppenflag ändern
ged_elemlaychg	GED Elementlage ändern
ged_elemmirrchg	GED Elementspiegelung ändern
ged_elemposchg	GED Elementposition ändern
ged_elemsizechg	GED Elementgröße ändern
ged_getautocornins	GED Modus für automatische Eckpunktgenerierung abfragen
ged_getdblpar	GED Doubleparameter abfragen
ged_getdrcmarkmode	GED DRC Fehleranzeigemodus abfragen
ged_getdrcstatus	GED DRC Vollständigkeitsstatus abfragen
ged_getgroupdata	GED Gruppenplatzierungsdaten abfragen
ged_gethighlnet	GED Netz Highlightmodus/Farbe abfragen
ged_getinputdata	GED Eingabedaten abfragen
ged_getintpar	GED Integerparameter abfragen
ged_getlaydefmode	GED Lagendefaultmodus abfragen
ged_getlayerdefault	GED Defaultlage abfragen
ged_getmincon	GED Mincon-Funktion abfragen
ged_getpathwidth	GED Bahnenstandardbreiten abfragen
ged_getpickmode	GED Elementpickmodus abfragen
ged_getpickpreflay	GED Vorzugslage abfragen
ged_getpowlayerrcnt	GED Versorgungslagenfehleranzahl abfragen

ged_getsegmovmode	GED Leiterbahnsegmentbewegungsmodus abfragen
ged_getstrpar	GED Stringparameter abfragen
ged_getviaoptmode	GED Leiterbahnviaoptimierungsmodus abfragen
ged_getwidedraw	GED Breitendarstellung abfragen
ged_groupselect	GED Gruppenselektion
ged_highlnet	GED Netz Highlightmodus/Farbe setzen
ged_layergrpchg	GED Gruppenselektion nach Lage
ged_partaltmacro	GED Bauteilgehäusetyp ändern
ged_partnamechg	GED Bauteilname ändern
ged_pickanyelem	Beliebiges GED Element selektieren
ged_pickelem	GED Element selektieren
ged_setautocornins	GED Modus für automatische Eckpunktgenerierung setzen
ged_setdblpar	GED Doubleparameter setzen
ged_setdrcmarkmode	GED DRC Fehleranzeigemodus setzen
ged_setintpar	GED Integerparameter setzen
ged_setlaydefmode	GED Lagendefaultmodus setzen
ged_setlayerdefault	GED Defaultlage setzen
ged_setmincon	GED Mincon-Funktion setzen
ged_setnetattrib	GED Netzattribut setzen
ged_setpathwidth	GED Bahnenstandardbreiten setzen
ged_setpickelem	GED Defaultpickelement setzen
ged_setpickmode	GED Elementpickmodus setzen
ged_setpickpreflay	GED Vorzugslage setzen
ged_setplantoplay	GED oberste Lage setzen
ged_setsegmovmode	GED Leiterbahnsegmentbewegungsmodus setzen
ged_setstrpar	GED Stringparameter setzen
ged_setviaoptmode	GED Leiterbahnviaoptimierungsmodus setzen
ged_setwidedraw	GED Breitendarstellung setzen
ged_storedrill	GED Bohrung platzieren
ged_storepart	GED Bauteil platzieren
ged_storepath	GED Bahn platzieren
ged_storepoly	GED Fläche platzieren
ged_storetext	GED Text platzieren
ged_storeuref	GED Via bzw. Pad platzieren

C.1.6 Autorouter Systemfunktionen (AR)

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp AR zugeordnet, d.h. diese Funktionen können im **Autorouter** aufgerufen werden:

ar_asklayer	Autorouter Lagenauswahl
ar_delelem	Autorouter Element löschen
ar_drawelem	Autorouter Elementanzeige aktualisieren
ar_elemangchg	Autorouter Elementwinkel ändern
ar_elemfixchg	Autorouter Element fixiert-Flag ändern
ar_elemmirrchg	Autorouter Elementspiegelung ändern
ar_elemposchg	Autorouter Elementposition ändern
ar_elemsizechg	Autorouter Elementgröße ändern
ar_getdblpar	Autorouter Doubleparameter abfragen
ar_getintpar	Autorouter Integerparameter abfragen
ar_getmincon	Autorouter Mincon-Funktion abfragen
ar_getpickpreflay	Autorouter Vorzugslage abfragen
ar_getstrpar	Autorouter Stringparameter abfragen
ar_getwidedraw	Autorouter Breitendarstellung abfragen
ar_highlnet	Autorouter Highlight Netz ein/aus
ar_partnamechg	Autorouter Bauteilname in Netzliste ändern
ar_pickelem	Autorouter Element mit Maus selektieren
ar_setdblpar	Autorouter Doubleparameter setzen
ar_setintpar	Autorouter Integerparameter setzen
ar_setmincon	Autorouter Mincon-Funktion setzen
ar_setnetattrib	Autorouter Netzattribut setzen
ar_setpickpreflay	Autorouter Vorzugslage setzen
ar_setplantoplay	Autorouter oberste Lage setzen
ar_setstrpar	Autorouter Stringparameter setzen
ar_setwidedraw	Autorouter Breitendarstellung setzen
ar_storepart	Autorouter Bauteil platzieren
ar_storepath	Autorouter Bahn platzieren
ar_storeuref	Autorouter Via bzw. Pad platzieren

C.1.7 CAM-Prozessor Systemfunktionen (CAM)

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp CAM zugeordnet, d.h. diese Funktionen können im **CAM-Prozessor** aufgerufen werden:

cam_askplotlayer	CAM Plotlagenauswahl
cam_getdblpar	CAM Doubleparameter abfragen
cam_getdrlaccuracy	CAM Bohrwerkzeugtoleranz abfragen
cam_getgenpltparam	CAM allgemeine Plotparameter abfragen
cam_getgerberapt	CAM Gerberblendendefinition abfragen
cam_getgerberparam	CAM Gerber-Parameter abfragen
cam_gethpglparam	CAM HP-GL-Plotparameter abfragen
cam_getintpar	CAM Integerparameter abfragen
cam_getplotlaycode	CAM Plotlagencode abfragen
cam_getpowpltparam	CAM Versorgungslagen-Parameter abfragen
cam_getwidedraw	CAM Breitendarstellung abfragen
cam_plotgerber	CAM Gerber-Ausgabe
cam_plothpgl	CAM HP-GL-Ausgabe
cam_setdblpar	CAM Doubleparameter setzen
cam_setdrlaccuracy	CAM Bohrwerkzeugtoleranz setzen
cam_setgenpltparam	CAM allgemeine Plotparameter setzen
cam_setgerberapt	CAM Gerberblende definieren
cam_setintpar	CAM Integerparameter setzen
cam_setplotlaycode	CAM Plotlagencode setzen
cam_setpowpltparam	CAM Versorgungslagen-Parameter setzen
cam_setwidedraw	CAM Breitendarstellung setzen

C.1.8 CAM-View Systemfunktionen (CV)

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp CV zugeordnet, d.h. diese Funktionen können im **CAM-View**-Modul aufgerufen werden:

cv_aptgetcolor	CAM-View Blendenfarbe abfragen
cv_aptsetcolor	CAM-View Blendenfarbe setzen
cv_deldataset	CAM-View Datensatz löschen
cv_getdblpar	CAM-View Doubleparameter abfragen
cv_getintpar	CAM-View Integerparameter abfragen
cv_movedataset	CAM-View Datensatz verschieben
cv_setdblpar	CAM-View Doubleparameter setzen
cv_setintpar	CAM-View Integerparameter setzen

C.1.9 IC Design Systemfunktionen (ICD)

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp ICD zugeordnet, d.h. diese Funktionen können im **Chipeditor** aufgerufen werden:

icd_altpinlay	IC Design Setup Alternativpinlayer
icd_cellconlay	IC Design Setup Lage interne Zellverbindungen
icd_cellscan	IC Design Setup DRC auf Zellebene
icd_cellshr	IC Design Setup Zellsperrflächenoffset
icd_ciflayname	IC Design Setup CIF-Ausgabelage abfragen
icd_cstdsiz	IC Design Setup Standardzellenhöhe abfragen
icd_defelemname	IC Design Setup default Elementname
icd_deflibname	IC Design Setup default Bibliothek
icd_drcarc	IC Design Setup DRC Kreisbögen abfragen
icd_drcgrid	IC Design Setup DRC Raster abfragen
icd_drclaymode	IC Design Setup DRC Lagenberücksichtigung
icd_drcmaxpar	IC Design Setup DRC Parallelcheck abfragen
icd_drcminwidth	IC Design Setup Lage minimale Strukturgröße
icd_drcrect	IC Design Setup DRC Orthogonalcheck abfragen
icd_ecnlaymode	IC Design Setup Lagenconnectivity abfragen
icd_findconpart	IC Design Bauteil in Netzliste suchen
icd_findconpartpin	IC Design Bauteilpin in Netzliste suchen
icd_findcontree	IC Design Netz in Netzliste suchen
icd_getrulecnt	IC Design-Element Regelanzahl abfragen
icd_getrulename	IC Design-Element Regelname abfragen
icd_gettreeidx	IC Design Netznummer in Netzliste suchen
icd_grpdisplay	IC Design Setup Gruppenlage abfragen
icd_lastfigelem	Zuletzt modifiziertes IC Design Element ermitteln
icd_maccoords	IC Design Makrokoordinaten abfragen
icd_nrefsearch	IC Design Name auf Plan suchen
icd_outlinelay	IC Design Setup Zellumrandung Lage abfragen
icd_pindist	IC Design Setup Pinaussparung abfragen
icd_plcxgrid	IC Design Setup Platzierungsraster abfragen
icd_plcxoffset	IC Design Setup Platzierungsoffset abfragen
icd_routcellcnt	IC Design Setup Anzahl Stromversorgungszellen
icd_routcellname	IC Design Setup Name Stromversorgungszelle
icd_ruleerr	Regelsystem Fehlerstatus abfragen

icd_rulefigatt	Regelzuweisung an Figurenelement
icd_rulefigdet	Regelzuweisungen von Figurenelement lösen
icd_ruleplanatt	Regelzuweisung an aktuell geladenes Element
icd_ruleplandet	Regelzuweisungen von aktuell geladenem Element lösen
icd_rulequery	IC Design-Element Regelabfrage durchführen
icd_scanall	IC Design Scan über alle Elemente
icd_scanfelem	IC Design Scan über Figurenelement
icd_scanpool	IC Design Scan über Poolelement
icd_stdlayname	IC Design Setup Standardlayer Name abfragen
icd_stdpinlay	IC Design Setup Standardpinlayer
icd_vecttext	IC Design Text vektorisieren

C.1.10 Chip Editor Systemfunktionen (CED)

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp CED zugeordnet, d.h. diese Funktionen können im **Chipeditor** aufgerufen werden:

ced_asklayer	CED Lagenauswahl
ced_delelem	CED Element löschen
ced_drawelem	CED Elementanzeige aktualisieren
ced_elemangchg	CED Elementwinkel ändern
ced_elemfixchg	CED Element fixiert-Flag ändern
ced_elemgrpchg	CED Element Gruppenflag ändern
ced_elemlaychg	CED Elementlage ändern
ced_elemmirrchg	CED Elementspiegelung ändern
ced_elemposchg	CED Elementposition ändern
ced_elemsizechg	CED Elementgröße ändern
ced_getlaydispmode	CED Lagenanzeigemodus abfragen
ced_getmincon	CED Mincon-Funktion abfragen
ced_getpathwidth	CED Bahnenstandardbreiten abfragen
ced_getpickpreflay	CED Vorzugslage abfragen
ced_getwidedraw	CED Breitendarstellung abfragen
ced_groupselect	CED Gruppenselektion
ced_highlnet	CED Highlight Netz ein/aus
ced_layergrpchg	CED Gruppenselektion nach Lage
ced_partaltmacro	CED Bauteilzellentyp ändern
ced_partnamechg	CED Bauteilname in Netzliste ändern
ced_pickelem	CED Element selektieren
ced_setlaydispmode	CED Lagenanzeigemodus setzen
ced_setmincon	CED Mincon-Funktion setzen
ced_setpathwidth	CED Bahnenstandardbreiten setzen
ced_setpickpreflay	CED Vorzugslage setzen
ced_setwidedraw	CED Breitendarstellung setzen
ced_storepart	CED Bauteil platzieren
ced_storepath	CED Bahn platzieren
ced_storepoly	CED Fläche platzieren
ced_storetext	CED Text platzieren
ced_storeuref	CED Via bzw. Subbauteil platzieren

C.2 Standard-Systemfunktionen

In diesem Abschnitt werden (in alphabetischer Reihenfolge) die in der **Bartels User Language** definierten Standard-Systemfunktionen beschrieben. Beachten Sie bitte die Konventionen zur Funktionsbeschreibung in Anhang C.1.

abs - Absolutwert eines Ganzzahlwertes (STD)

Synopsis

```
int abs(                        // Berechnungsergebnis
      int;                      // Eingabewert
      );
```

Beschreibung

Der Rückgabewert der Funktion **abs** entspricht dem Absolutwert des übergebenen Ganzzahlwertes.

acos - Arcuscosinus berechnen (STD)

Synopsis

```
double acos(                    // Berechnungsergebnis (STD3)
      double [-1.0,1.0];        // Cosinus-Wert
      );
```

Beschreibung

Der Rückgabewert der Funktion **acos** entspricht dem Arcuscosinus des übergebenen Gleitkommawertes. Der Winkel wird in Bogenmaßeinheiten angegeben.

angclass - Winkelklassifizierung (STD)

Synopsis

```
int angclass(                   // Winkelklasse
      double;                   // Winkel (STD3)
      );
```

Beschreibung

Der Rückgabewert der Funktion **angclass** gibt die Winkelklasse eines Winkels an. Mögliche Klassenwerte sind 0 für 0 Grad Winkel, 1 für einen 90 Grad Winkel, 2 für einen 180 Grad Winkel, 3 für einen 270 Grad Winkel und (-1) für andere Winkel. Der übergebene Gleitkommawert wird als Bogenmaßangabe interpretiert.

arylength - Arraylänge bestimmen (STD)

Synopsis

```
int arylength(                  // Feldlänge
      void;                     // Array beliebigen Typs
      );
```

Beschreibung

Der Rückgabewert der Funktion **arylength** entspricht der Länge des übergebenen Arrays.

asin - Arcussinus berechnen (STD)

Synopsis

```
double asin(                    // Berechnungsergebnis (STD3)
      double [-1.0,1.0];        // Sinus-Wert
      );
```

Beschreibung

Der Rückgabewert der Funktion **asin** entspricht dem Arcussinus des übergebenen Gleitkommawertes. Der Winkel wird in Bogenmaßeinheiten angegeben.

askcoord - Benutzereingabe X-/Y-Koordinatenwerte (STD)

Synopsis

```
int askcoord(                 // Status
      & double;               // Rückgabe X-Koordinate (STD2)
      & double;               // Rückgabe Y-Koordinate (STD2)
      int [0,1];              // Eingabemodus:
                              //    0 = Koordinaten relativ zu letzter Position
                              //    1 = Koordinaten absolut
      );
```

Beschreibung

Die Funktion **askcoord** aktiviert einen Dialog zur Abfrage von X- und Y-Koordinatenwerten. Der Eingabemodus gibt an, ob Relativkoordinaten (Sprung relativ) oder Absolutkoordinaten (Sprung absolut) abgefragt werden sollen. Die eingegebenen Koordinatenwerte werden über die ersten beiden Funktionsparameter an den Aufrufer zurückgegeben. Bei der Abfrage von Absolutkoordinaten fungieren diese Parameter auch als Eingabeparameter für voreingestellte Koordinatenwerte. Der Funktionsrückgabewert ist Null bei erfolgreicher Koordinateneingabe oder ungleich Null wenn die Koordinateneingabe abgebrochen wurde.

askdbl - Benutzereingabe Gleitkommazahlwert (STD)

Synopsis

```
int askdbl(                   // Status
      & double;               // Rückgabe Gleitkommawert
      string;                 // Eingabeaufforderung
      int;                    // Maximale Eingabelänge
      );
```

Beschreibung

Die Funktion **askdbl** fordert den Benutzer in der Eingabezeile mit der übergebenen Promptzeichenkette zur Eingabe eines Gleitkommawertes auf. Der vom Benutzer eingegebene Wert wird im Rückgabeparameter zurückgegeben. Ein Funktionsrückgabewert ungleich Null gibt an, dass kein gültiger Gleitkommawert eingegeben wurde.

askdist - Benutzereingabe Distanzwert (STD)

Synopsis

```
int askdist(                  // Status:
                              //    -1 = Eingabe ungültig/abgebrochen
                              //     0 = Gültige Distanzwerteingabe
                              //     1 = Gültige Distanzwerteingabe,
                              //          Eckpunktschaltfläche gedrückt
      & double;               // Rückgabe Distanzwert
      string;                 // Eingabeaufforderung
      int [0,15];             // Eingabekontrolle:
                              //    1 = Negative Werte zulässig
                              //    2 = Kreisbögen zulässig
                              //    4 = Eingabeaufferdoerung für Zeichenketten
                              //    8 = Schaltfläche für runde Ecken
      );
```

Beschreibung

Die Funktion **askdist** fordert den Benutzer in der Eingabezeile mit der übergebenen Promptzeichenkette zur Eingabe eines Distanzwertes auf. Der vom Benutzer eingegebene Wert wird im Default-User-Einheitensystem interpretiert und im Rückgabeparameter in der Einheit Meter zurückgegeben. Der dritte Parameter dient der Konfiguration der zulässigen Eingaben. Der Funktionsrückgabewert ergibt sich zu Null wenn ein gültiger Distanzwert eingegeben wurde, 1 bei erfolgreicher Distanzwerteingabe mit gedrückter Eckpunktschaltfläche, oder (-1) wenn die kein gütiger Wert eingegeben bzw. die Eingabe abgebrochen wurde.

askint - Benutzereingabe Ganzzahlwert (STD)

Synopsis

```
int askint(                     // Status
      & int;                    // Rückgabe Ganzzahlwert
      string;                   // Eingabeaufforderung
      int;                      // Maximale Eingabelänge
      );
```

Beschreibung

Die Funktion **askint** fordert den Benutzer in der Eingabezeile mit der übergebenen Promptzeichenkette zur Eingabe eines Ganzzahlwertes auf. Der vom Benutzer eingegebene Wert wird im Rückgabeparameter zurückgegeben. Ein Funktionsrückgabewert ungleich Null gibt an, dass keine gültiger Ganzzahlwert eingegeben wurde.

askstr - Benutzereingabe Zeichenkette (STD)

Synopsis

```
string askstr(                  // Zeichenkette
      string;                   // Eingabeaufforderung
      int;                      // Maximale Eingabelänge
      );
```

Beschreibung

Die Funktion **askstr** fordert den Benutzer in der Eingabezeile mit der übergebenen Promptzeichenkette zur Eingabe einer Zeichenkette auf. Der Rückgabewert dieser Funktion entspricht der vom Benutzer eingegebenen Zeichenkette.

atan - Arcustangens berechnen (STD)

Synopsis

```
double atan(                    // Berechnungsergebnis
      double;                   // Winkel (STD3)
      );
```

Beschreibung

Der Rückgabewert der Funktion **atan** entspricht dem Arcustangens des übergebenen Gleitkommawertes. Der Winkel wird in Bogenmaßeinheiten angegeben.

atan2 - Arcustangens eines punktbestimmten Winkels (STD)

Synopsis

```
double atan2(                   // Berechnungsergebnis (STD3)
      double;                   // Y-Koordinate
      double;                   // X-Koordinate
      );
```

Beschreibung

Der Rückgabewert der Funktion **atan2** entspricht dem Winkel, den die Ursprungshalbgerade durch den angegebenen Punkt mit der positiven X-Achse einschließt. Der Winkel wird in Bogenmaßeinheiten angegeben.

atof - Umwandlung Zeichenkette in Gleitkommazahlwert (STD)

Synopsis

```
double atof(                    // Gleitkommawert
      string;                   // Zeichenkette
      );
```

Beschreibung

Der Rückgabewert der Funktion **atof** entspricht dem Gleitkommawert, der in der übergebenen Zeichenkette in ASCII-Darstellung angegeben ist.

atoi - Umwandlung Zeichenkette in Ganzzahlwert (STD)

Synopsis

```
int atoi(                      // Ganzzahlwert
      string;                  // Zeichenkette
      );
```

Beschreibung

Der Rückgabewert der Funktion **atoi** entspricht dem Ganzzahlwert, der in der übergebenen Zeichenkette in ASCII-Darstellung angegeben ist.

bae_askddbename - DDB-Elementnamensabfrage (STD)

! prefix save else load, empty if std. message

Synopsis

```
int bae_askddbename(           // Status
      & string;                // Rückgabe Elementname
      string;                  // Dateiname
      int ]0,[;                // Elementklasse (STD1)
      string;                  // Eingabeaufforderung
                               //    Leerstring: Standarddateidialog
                               //    !-Präfix: Dateisicherungsdialog
                               //    sonst: Dateiladedialog
      );
```

Beschreibung

Die Funktion **bae_askddbename** erlaubt dem Benutzer einen DDB-Datei Elementnamen wahlweise mit der Maus in einem Popupauswahlfenster aus der Liste der vorhandenen Elemente zu selektieren oder per Tastatur in der Eingabezeile einzugeben. Wird für die Eingabeaufforderung ein Leerstring angegeben, so wird der Standardprompt für DDB-Elementnamensabfragen verwendet. Über den ersten Parameter kann ein Name übergeben werden, der in der Abfrage als Vorauswahl erscheint, d.h. wenn keine Vorauswahl möglich bzw. erlaubt ist, dann ist ein Leerstring zu übergeben. Der Rückgabewert ist ungleich Null, wenn kein gültiges Element gewählt wurde.

bae_askddbfname - DDB-Dateinamensabfrage (STD)

Synopsis

```
int bae_askddbfname(           // Status
      & string;                // Rückgabe Dateiname
      int [0,3];               // Modus Dateinamensdialog (Bitmuster):
                               //    1 = Prüfen, ob Datei existiert
                               //    2 = angegebener Dateiname ist Default
      string;                  // Eingabeaufforderung
                               //    Leerstring: Standarddateidialog
                               //    !-Präfix: Dateisicherungsdialog
                               //    sonst: Dateiladedialog
      );
```

Beschreibung

Die Funktion **bae_askddbfname** erlaubt dem Benutzer einen DDB-Dateinamen wahlweise mit der Maus in einem Popupauswahlfenster aus der Liste der vorhandenen DDB-Dateien zu selektieren oder per Tastatur in der Eingabezeile einzugeben. Wird für die Eingabeaufforderung ein Leerstring angegeben, dann wird der BAE-Standardprompt für DDB-Dateinamensabfragen verwendet. Ist das erste Zeichen der Eingabeaufforderung ein Ausrufezeichen (`!`), dann wird dieses ausgeblendet, und unter Windows wird im Dateiabfragedialog anstelle des Bestätigungsbuttons Oeffnen der Button Speichern angezeigt. Der Rückgabewert ist ungleich Null, wenn kein gültiger DDB-Dateiname gewählt wurde bzw. die Existenzprüfung auf eins gesetzt ist und die angegebene Datei nicht existiert.

bae_askdirname - Dateiverzeichnisnamensabfrage (STD)

Synopsis

```
int bae_askdirname(            // Status
      & string;                // Rückgabe Dateiverzeichnisname
      string;                  // Startverzeichnispfad
      string;                  // Eingabeaufforderung
      );
```

Beschreibung

Die Funktion **bae_askdirname** erlaubt dem Benutzer einen Dateiverzeichnisnamen wahlweise mit der Maus in einem Popupauswahlfenster aus der Liste der vorhandenen Verzeichnisse zu selektieren oder per Tastatur in der Eingabezeile einzugeben. Der Startverzeichnispfad gibt das Verzeichnis an ab dem im Popupauswahlfenster Unterverzeichnisse angezeigt werden. Wird für die Eingabeaufforderung ein Leerstring angegeben, so wird der Standardprompt für Dateiverzeichnisabfragen verwendet. Der Rückgabewert ist ungleich Null, wenn kein gültiger Verzeichnisname gewählt wurde.

bae_askfilename - Dateinamensabfrage (STD)

Synopsis

```
int bae_askfilename(           // Status
      & string;                // Rückgabe Dateiname
      string;                  // Dateinamensendung
      string;                  // Eingabeaufforderung
                               //    Leerstring: Standarddateidialog
                               //    !-Präfix: Dateisicherungsdialog
                               //    sonst: Dateiladedialog
      );
```

Beschreibung

Die Funktion **bae_askfilename** aktiviert einen Dialog zur Dateinamensabfrage. Über den Parameter für die Dateinamensendung kann ein Dateinamensfilter gesetzt werden. Ein Leerstring bewirkt die Anzeige aller verfügbaren Dateien. Bei Angabe einer Dateinamenserweiterung (z.B. **`.ddb`**, **`.dat`**, **`.txt`**, **`.`**, **`.*`**, usw.) werden nur die Dateien mit der angegebenen Endung angezeigt. Bei Eingabe eines Minuszeichens (**`-`**) für die Dateinamenserweiterung werden alle Dateien mit BAE-System- bzw. Datendateiendungen (**`.ass`**, **`.con`**, **`.ddb`**, **`.def`**, **`.exe`**, **`.fre`**, **`.ulc`** und **`.usf`**) aus dem Dateinamensmenü ausgeblendet (dieser Mechanismus eignet sich insbesondere für Ausgabe- bzw. Plotdateiabfragen, bei welchen keine System-, Bibliotheks- oder Projektdateien selektiert werden dürfen). Wird für die Eingabeaufforderung ein Leerstring angegeben, dann wird der BAE-Standardprompt für Dateinamensabfragen verwendet. Ist das erste Zeichen der Eingabeaufforderung ein Ausrufezeichen (**`!`**), dann wird dieses ausgeblendet, und unter Windows wird im Dateiabfragedialog anstelle des Bestätigungsbuttons Oeffnen der Button Speichern angezeigt. Der Rückgabewert ist ungleich Null, wenn kein gültiger Dateiname gewählt wurde.

bae_askmenu - BAE Menüabfrage (STD)

Synopsis

```
int bae_askmenu(               // Rückgabe selektierter Menüeintrag, (0..49)
                               //  oder (-1) bei Abbruch der Menüauswahl
      int [1,50];              // Anzahl Menüeinträge
      string;                  // Menütext erster Menüeintrag
      []                       // Weitere Menütexte
      );
```

Beschreibung

Die Funktion **bae_askmenu** ermöglicht die Definition und Aktivierung eines anwenderspezifischen Menüs mit mausselektierbaren Menüeinträgen. Der Rückgabewert gibt den vom Benutzer gewählten Menüeintrag an oder ergibt sich zu (-1), wenn die Menüauswahl abgebrochen wurde. Die Nummerierung der Menüeinträge beginnt bei Null.

Siehe auch

Funktion **bae_defmenusel**.

bae_askname - BAE Dialog zur Namensauswahl aktivieren (STD)

Synopsis

```
int bae_askname(                // Status
      & string;                 // Rückgabe selektierter Name
      string;                   // Eingabeaufforderung (oder Leerstring)
      int;                      // Maximale Eingabestringlänge
      );
```

Beschreibung

Die Funktion **bae_askname** aktiviert einen Dialog zur Selektion eines Namens aus der aktuell mit **bae_nameadd** definierten Namensliste. Über den zweiten Funktionsparameter kann eine anwendungsspezifische Eingabeaufforderung angegeben werden. Wird hierfür ein Leerstring angegeben, dann benutzt **bae_askname** einen vordefinierten Standardprompt. Der dritte Funktionsparameter definiert die maximal zulässige Stringlänge für die Benutzereingabe. Bei erfolgter Namensauswahl wird der selektierte Name im ersten Funktionsparameter zurückgegeben. Der Funktionsrückgabewert ergibt ist Null bei erfolgreicher Namensselektion oder ungleich Null wenn die Funktion ohne Namensselektion abgebrochen wurde.

Siehe auch

Funktionen **bae_nameadd**, **bae_nameclr**, **bae_nameget**.

bae_asksymname - BAE DDB-Bibliothekselementabfrage (STD)

Synopsis

```
int bae_asksymname(             // Status
      & string;                 // Rückgabe Elementname
      & string;                 // Rückgabe DDB-Bibliotheksdateiname
      int ]0,[;                 // Datenbankklasse (STD1)
      string;                   // Bibliotheksverzeichnis
      string;                   // Standard-Bibliotheksdateiname
      string;                   // Standard-Symbol-/Elementname
      );
```

Beschreibung

Die Funktion **bae_asksymname** aktiviert einen Dialog zur Auswahl eines Bibliothekselements der angegeben Datenbankklasse aus einer selektierbaren Bibliotheksdatei. Der Funktionsrückgabewert ist Null bei erfolgreicher Auswahl oder ungleich Null wenn der Anwender den Dialog ohne gültige Elementauswahl beendet.

bae_callmenu - BAE Menüfunktion aufrufen (STD)

Synopsis

```
int bae_callmenu(               // Status
      int [0,9999];             // Menüfunktion (STD4)
      );
```

Beschreibung

Die Funktion **bae_callmenu** aktiviert die angegebene Menüfunktion. Der Rückgabewert ist ungleich Null, wenn bei der Ausführung der Menüfunktion ein Fehler aufgetreten ist oder eine ungültige bzw. unbekannte Menüfunktion spezifiziert wurde.

bae_charsize - BAE Text-/Zeichengröße abfragen (STD)

Synopsis

```
void bae_charsize(
      & double;                 // Rückgabe Zeichenbreite (Pixelanzahl)
      & double;                 // Rückgabe Zeichenhöhe (Pixelanzahl)
      );
```

Beschreibung

Die Funktion **bae_charsize** gibt in den beiden Parametern die aktuell im **AutoEngineer** eingestellte Text- bzw. Zeichengröße, d.h. die Pixelanzahl pro Zeichen in X- bzw. Y-Richtung zurück.

bae_cleardistpoly - Internes BAE Distanzabfragepolygon löschen (STD)

Synopsis

```
void bae_cleardistpoly(
      );
```

Beschreibung

Die Funktion **bae_cleardistpoly** löscht das mit **bae_storedistpoly** erzeugte Distanzabfragepolygon.

Siehe auch

Funktion **bae_storedistpoly**.

bae_clearpoints - BAE Polygonpunktliste löschen (STD)

Synopsis

```
void bae_clearpoints(
      );
```

Beschreibung

Die Funktion **bae_clearpoints** löscht die zum Aufbau von Polygonen verwendete interne Punktliste. Dies ist prinzipiell vor dem ersten Aufruf von **bae_storepoint** zum Aufbau einer Punktliste notwendig um evtl. vorhandene alte Punkte zu löschen.

Siehe auch

Funktionen **bae_getpolyrange**, **bae_storedistpoly**, **bae_storepoint**.

bae_clriactqueue - BAE Interaktionsvorgaben löschen (STD)

Synopsis

```
void bae_clriactqueue(
      );
```

Beschreibung

Die Funktion **bae_clriactqueue** löscht die in der Interaktionsqueue gespeicherten Vorgaben. Die Interaktionsqueue dient dazu, die Benutzereingaben für die über die Funktion **bae_callmenu** aufgerufenen Menüfunktionen ganz oder teilweise vorzugeben.

bae_crossarcarc - Schnittpunkt(e) zwischen Kreisbögen bestimmen (STD)

Synopsis

```
int bae_crossarcarc(          // Anzahl Schnittpunkte
      double;                 // Kreisbogen 1 Start X-Koordinate (STD2)
      double;                 // Kreisbogen 1 Start Y-Koordinate (STD2)
      double;                 // Kreisbogen 1 Mittelpunkt X-Koordinate (STD2)
      double;                 // Kreisbogen 1 Mittelpunkt Y-Koordinate (STD2)
      int [1,2];              // Kreisbogen 1 Mittelpunkt Typ (STD15)
      double;                 // Kreisbogen 1 Endpunkt X-Koordinate (STD2)
      double;                 // Kreisbogen 1 Endpunkt Y-Koordinate (STD2)
      double;                 // Kreisbogen 2 Startpunkt X-Koordinate (STD2)
      double;                 // Kreisbogen 2 Startpunkt Y-Koordinate (STD2)
      double;                 // Kreisbogen 2 Mittelpunkt X-Koordinate (STD2)
      double;                 // Kreisbogen 2 Mittelpunkt Y-Koordinate (STD2)
      int [1,2];              // Kreisbogen 2 Mittelpunkt Typ (STD15)
      double;                 // Kreisbogen 2 Endpunkt X-Koordinate (STD2)
      double;                 // Kreisbogen 2 Endpunkt Y-Koordinate (STD2)
      & double;               // Schnittpunkt 1 X-Koordinate (STD2)
      & double;               // Schnittpunkt 1 Y-Koordinate (STD2)
      & double;               // Schnittpunkt 2 X-Koordinate (STD2)
      & double;               // Schnittpunkt 2 Y-Koordinate (STD2)
      );
```

Beschreibung

Die Funktion **bae_crossarcarc** führt eine Schnittpunktbestimmung für die beiden angegebenen Kreisbögen durch. Der Funktionsrückgabewert gibt die Anzahl der Schnittpunkte an (0, 1 oder 2). Die ermittelten Schnittpunktkoordinaten werden ggf. in den entsprechenden Funktionsparametern zurückgegeben.

Siehe auch

Funktionen **bae_crosslineline**, **bae_crosslinepoly**, **bae_crosssegarc**, **bae_crosssegseg**.

bae_crosslineline - Schnittpunkt zwischen Liniensegmenten mit Breite bestimmen (STD)

Synopsis

```
int bae_crosslineline(        // Schnittflag
      double;                 // Linie 1 Start X-Koordinate (STD2)
      double;                 // Linie 1 Start Y-Koordinate (STD2)
      double;                 // Linie 1 Endpunkt X-Koordinate (STD2)
      double;                 // Linie 1 Endpunkt Y-Koordinate (STD2)
      double ]0.0,[;          // Linie 1 Breite (STD2)
      double;                 // Linie 2 Start X-Koordinate (STD2)
      double;                 // Linie 2 Start Y-Koordinate (STD2)
      double;                 // Linie 2 Endpunkt X-Koordinate (STD2)
      double;                 // Linie 2 Endpunkt Y-Koordinate (STD2)
      double ]0.0,[;          // Linie 2 Breite (STD2)
      );
```

Beschreibung

Die Funktion **bae_crosslineline** prüft ob sich die beiden angegebenen Liniensegmente schneiden. Der Funktionsrückgabewert ist 1 wenn sich die Segmente schneiden oder Null andernfalls.

Siehe auch

Funktionen **bae_crossarcarc**, **bae_crosslinepoly**, **bae_crosssegarc**, **bae_crosssegseg**.

bae_crosslinepoly - Schnittpunkt zwischen Liniensegment mit Breite und Polygon bestimmen (STD)

Synopsis

```
int bae_crosslinepoly(         // Schnittflag
      double;                  // Linie Start X-Koordinate (STD2)
      double;                  // Linie Start Y-Koordinate (STD2)
      double;                  // Linie Endpunkt X-Koordinate (STD2)
      double;                  // Linie Endpunkt Y-Koordinate (STD2)
      double ]0.0,[;           // Linie Breite (STD2)
      );
```

Beschreibung

Die Funktion **bae_crosslinepoly** prüft ob das angegebene Liniensegmente das mit **bae_storepoint** aktuell erzeugte Polygon schneidet. schneiden. Der Funktionsrückgabewert ist 1 wenn sich das Liniensegmente mit dem Polygon schneidet oder Null andernfalls.

Siehe auch

Funktionen **bae_crossarcarc**, **bae_crosslineline**, **bae_crosssegarc**, **bae_crosssegseg**, **bae_storepoint**.

bae_crosssegarc - Schnittpunkt zwischen Liniensegment und Kreisbogen bestimmen (STD)

Synopsis

```
int bae_crosssegarc(           // Anzahl Schnittpunkte
      double;                  // Segment Start X-Koordinate (STD2)
      double;                  // Segment Start Y-Koordinate (STD2)
      double;                  // Segment Endpunkt X-Koordinate (STD2)
      double;                  // Segment Endpunkt Y-Koordinate (STD2)
      double;                  // Kreisbogen Start X-Koordinate (STD2)
      double;                  // Kreisbogen Start Y-Koordinate (STD2)
      double;                  // Kreisbogen Mittelpunkt X-Koordinate (STD2)
      double;                  // Kreisbogen Mittelpunkt Y-Koordinate (STD2)
      int [1,2];               // Kreisbogen Mittelpunkt Typ (STD15)
      double;                  // Kreisbogen Endpunkt X-Koordinate (STD2)
      double;                  // Kreisbogen Endpunkt Y-Koordinate (STD2)
      int [0,1];               // Schnittpunktprioritätsflag
      & double;                // Schnittpunkt 1 X-Koordinate (STD2)
      & double;                // Schnittpunkt 1 Y-Koordinate (STD2)
      & double;                // Schnittpunkt 2 X-Koordinate (STD2)
      & double;                // Schnittpunkt 2 Y-Koordinate (STD2)
      );
```

Beschreibung

Die Funktion **bae_crosssegarc** führt eine Schnittpunktbestimmung für das angegebene Segment und den angegebenen Kreisbogen durch. Der Funktionsrückgabewert gibt die Anzahl der Schnittpunkte an (0, 1 oder 2). Die ermittelten Schnittpunktkoordinaten werden ggf. in den entsprechenden Funktionsparametern zurückgegeben.

Siehe auch

Funktionen **bae_crossarcarc**, **bae_crosslineline**, **bae_crosslinepoly**, **bae_crosssegseg**.

bae_crosssegseg - Schnittpunkt zwischen Liniensegmenten/Linien bestimmen (STD)

Synopsis

```
int bae_crosssegseg(          // Schnittflag
      int [0,1];              // Geradenvergleichsflag
      double;                 // Linie 1 Start X-Koordinate (STD2)
      double;                 // Linie 1 Start Y-Koordinate (STD2)
      double;                 // Linie 1 Endpunkt X-Koordinate (STD2)
      double;                 // Linie 1 Endpunkt Y-Koordinate (STD2)
      double;                 // Linie 2 Start X-Koordinate (STD2)
      double;                 // Linie 2 Start Y-Koordinate (STD2)
      double;                 // Linie 2 Endpunkt X-Koordinate (STD2)
      double;                 // Linie 2 Endpunkt Y-Koordinate (STD2)
      & double;               // Schnittpunkt X-Koordinate (STD2)
      & double;               // Schnittpunkt Y-Koordinate (STD2)
      );
```

Beschreibung

Die Funktion **bae_crosssegseg** prüft ob sich die beiden angegebenen Segmente bzw. Geraden schneiden. Der erste Funktionsparameter gibt an ob ein Geraden- oder ein Streckenvergleich durchgeführt werden soll. Der Funktionsrückgabewert ist 1 wenn ein Schnittpunkt ermittelt wurde oder Null andernfalls. Wenn ein Schnittpunkt ermittelt werden konnte, dann werden dessen Koordinaten über die letzten beiden Funktionsparameter zurückgegeben.

Siehe auch

Funktionen **bae_crossarcarc**, **bae_crosslineline**, **bae_crosslinepoly**, **bae_crosssegarc**.

bae_dashpolyline - Gestricheltes BAE Polygon vektorisieren (STD)

Synopsis

```
int bae_dashpolyline(          // Status
      int;                     // Polygonstrichelungsmodus:
                               //    0 = durchgezogene Linie (keine Strichelung)
                               //    1 = gestrichelte Linie
                               //    2 = gepunktete Linie
                               //    3 = gestrichelte/gepunktete Linie
      double ]0.0,[;           // Polygonbasisstrichlänge (STD2)
      double ]-0.5,0.5[;       // Relativer Polygonstrichabstand
      * int;                   // Polygonlinienscanfunktion
      * int;                   // Polygonbogenscanfunktion
      );
```

Beschreibung

Die Funktion **bae_dashpolyline** vektorisiert das zuvor mit **bae_storepoint** gespeicherte interne Polygon unter Berücksichtigung der spezifizierten Strichelungsparameter. Die Polygonlinienscanfunktion und die Polygonbogenscanfunktion werden jeweils automatisch für die im Polygon enthaltenen Linien bzw. Bögen aufgerufen. Der Funktionsrückgabewert ist Null bei erfolgreicher Vektorisierung oder ungleich Null im Fehlerfall bzw. wenn die Vektorisierung abgebrochen wurde.

Polygonlinienscanfunktion

```
int polylinescanfuncname(
      double xs,               // Linienstartpunkt X-Koordinate (STD2)
      double ys,               // Linienstartpunkt Y-Koordinate (STD2)
      double xe,               // Linienendpunkt X-Koordinate (STD2)
      double ye                // Linienendpunkt Y-Koordinate (STD2)
      )
{
      // Polygon line scan function statements
      :
      return(status);
}
```

Die Polygonlinienscanfunktion sollte den Wert Null zurückgeben wenn die Abarbeitung erfolgreich war. Im anderen Fall (bei Fehlern bzw. zum Abbruch des Scans) sollte ein Wert ungleich Null zurückgegeben werden.

Polygonbogenscanfunktion

```
int polylinescanfuncname(
      double xs,               // Bogenstartpunkt X-Koordinate (STD2)
      double ys,               // Bogenstartpunkt Y-Koordinate (STD2)
      double xe,               // Bogenendpunkt X-Koordinate (STD2)
      double ye                // Bogenendpunkt Y-Koordinate (STD2)
      double xc,               // Bogenmittelpunkt X-Koordinate (STD2)
      double yc                // Bogenmittelpunkt Y-Koordinate (STD2)
      int cwflag;              // Bogendrehrichtung:
                               //       0 = Bogen im Gegenuhrzeigersinn
                               //   sonst = Bogen im Uhrzeigersinn
      )
{
      // Polygon arc scan function statements
      :
      return(status);
}
```

Die Polygonbogenscanfunktion sollte den Wert Null zurückgeben wenn die Abarbeitung erfolgreich war. Im anderen Fall (bei Fehlern bzw. zum Abbruch des Scans) sollte ein Wert ungleich Null zurückgegeben werden.

Siehe auch

Funktionen **bae_clearpoints**, **bae_storepoint**.

bae_deffuncprog - BAE Funktionstaste programmieren (STD)

Synopsis

```
int bae_deffuncprog(            // Status
      int [1,128];              // Funktionstastennummer
      string;                   // User Language-Programmname oder
                                //  # gefolgt von Menünummer (STD4)
      );
```

Beschreibung

Die Funktion **bae_deffuncprog** ordnet einer Funktionstaste das namentlich angegebene **User Language**-Programm bzw. die numerisch spezifizierte BAE-Menüfunktion zu. Durch die Spezifikation eines Leerstrings für den Programmnamen kann die aktuelle Zuordnung zurückgesetzt werden. Der Rückgabewert ist ungleich Null, wenn ungültige Parameter übergeben wurden, oder wenn die zurückzusetzende Funktionstaste nicht belegt ist.

Siehe auch

Funktionen **bae_getfuncprog**, **bae_resetmenuprog**.

bae_defkeyprog - BAE Standardtaste programmieren (STD)

Synopsis

```
int bae_defkeyprog(             // Status
      int;                      // Tastenzeichen
      string;                   // User Language-Programmname oder
                                //  # gefolgt von Menünummer (STD4)
      );
```

Beschreibung

Die Funktion **bae_defkeyprog** ordnet einer Standardtaste das namentlich angegebene **User Language**-Programm bzw. die numerisch spezifizierte BAE-Menüfunktion zu. Durch die Spezifikation eines Leerstrings für den Programmnamen kann die aktuelle Zuordnung zurückgesetzt werden. Der Rückgabewert ist ungleich Null, wenn ungültige Parameter übergeben wurden, oder wenn die zurückzusetzende Standardtaste nicht belegt ist.

Siehe auch

Funktionen **bae_getkeyprog**, **bae_resetmenuprog**.

bae_defmenu - BAE Menüdefinition starten (STD)

Synopsis

```
int bae_defmenu(                // Status
      int [0,999];              // Menücode
      int [0,999];              // Menübereichscode:
                                //    1   = Hauptmenü
                                //    101 = erstes Untermenü
                                //    102 = zweites Untermenü
                                //    :   = : Untermenü
      );
```

Beschreibung

Die Funktion **bae_defmenu** startet die Definition eines Standardmenüs im aktuell aktiven BAE-Modul. Der Rückgabewert der Funktion **bae_defmenu** ist (-1), wenn ein Fehler aufgetreten ist oder Null andernfalls. Im Anschluss an den Aufruf der Funktion **bae_defmenu** sollten durch wiederholten Aufruf der Funktion **bae_defmenutext** die einzelnen Menüeinträge definiert werden. Die mit **bae_defmenu** eingeleitete Menüdefinition *muss* durch einen Aufruf der Funktion **bae_endmenu** beendet werden. Mit der Funktion **bae_resetmenuprog** können *sämtliche* Menübelegungen wieder zurückgesetzt werden.

Siehe auch

Funktionen **bae_defmenuprog**, **bae_defmenutext**, **bae_endmenu**, **bae_redefmenu**, **bae_resetmenuprog**.

bae_defmenuprog - BAE Menüfunktion programmieren (STD)

Synopsis

```
int bae_defmenuprog(          // Selectionscode or (-1) on error
      int [0,999];            // Menünummer
      int [0,99];             // Menüzeile
      string;                 // Menütext
      string;                 // User Language-Programmname oder
                              //  # gefolgt von Menünummer (STD4)
      int;                    // Menüeintrag Bearbeitungsschlüssel:
                              //    8000000h = immer aktivierbar
                              //    7FFFFFFh = für jeden Elementtyp
                              //               aktivierbar
                              //    sonstige = (kombinierter)
                              //               DDB-Klassen-Schlüssel
      );
```

Beschreibung

Die Funktion **bae_defmenuprog** ordnet einem Menüeintrag den spezifizierten Menütext sowie das namentlich angegebene **User Language**-Programm bzw. die numerisch spezifizierte BAE-Menüfunktion zu. Die Menünummer gibt die Nummer des Hauptmenüs an, die Menüzeile die Position im zugehörigen Submenü. Durch die Spezifikation eines Leerstrings für den Programmnamen kann die aktuelle Zuordnung zurückgesetzt werden. Der Bearbeitungsschlüssel dient der Konfiguration von sogenannten Ghostmenüs. Hierfür ist ein kodierter Integerwert einzutragen, wie er mit den Funktionen **bae_getclassbitfield** oder **bae_getmenubitfield** ermittelt bzw. definiert werden kann (im Zweifelsfall empfiehlt sich die Verwendung des Hexadezimalwertes 80000000h, um sicherzustellen, dass die Funktion immer aktivierbar ist). Der Rückgabewert der Funktion **bae_defmenuprog** ist ungleich Null, wenn ungültige Parameter übergeben wurden, oder wenn der zurückzusetzende Menüeintrag nicht belegt ist.

Siehe auch

Funktionen **bae_getclassbitfield**, **bae_getmenubitfield**, **bae_getmenuprog**, **bae_getmenutext**, **bae_redefmenu**, **bae_resetmenuprog**.

bae_defmenusel - BAE Menüabfrage Vorauswahl (STD)

Synopsis

```
void bae_defmenusel(
      int [-1,29];            // Menüindex
                              //  oder (-1) für Selektionstextspeicher
      );
```

Beschreibung

Die Funktion **bae_defmenusel** setzt die Vorauswahl bzw. Selektionsanzeige für den nächsten Aufruf von **bae_askmenu**. Damit lässt sich in Auswahlmenüs die aktuell selektierte Option kennzeichnen. In den BAE-Windowsversionen erfolgt die Selektionsmarkierung durch ein Häkchen, in den Motifversionen durch Grauunterlegung und in den DOS-Versionen bzw. Seitenmenükonfigurationen durch Vorauswahl des über den Funktionsparameter spezifizierten Menüpunkts. Die Anzeige bzw. Auswahl ist nur für den nächsten **bae_askmenu**-Aufruf gültig, d.h. nach dem **bae_askmenu**-Aufruf ist die Vorauswahl zurügesetzt und ist nach Bedarf mit **bae_defmenusel** neu zu setzen.

Siehe auch

Funktion **bae_askmenu**.

bae_defmenutext - BAE Menütext definieren (STD)

Synopsis

```
int bae_defmenutext(          // Status
      int [0,99];             // Menüzeile
      string;                 // Menütext
      int;                    // Menüeintrag Bearbeitungsschlüssel:
                              //    8000000h = immer aktivierbar
                              //    7FFFFFFh = für jeden Elementtyp
                              //                aktivierbar
                              //    sonstige = (kombinierter)
                              //                DDB-Klassen-Schlüssel
      );
```

Beschreibung

Die Funktion **bae_defmenutext** kann im Anschluss an den Aufruf einer der Funktionen **bae_defmenu** oder **bae_defselmenu** wiederholt ausgeführt werden, um an der jeweils angegebenen Menüzeile der aktuellen Menüdefinition den spezifizierten Menütext einzutragen. Ist das erste Zeichen des Menütexts ein Prozentzeichen (**%**), dann wird vor dem zu definierenden Pulldownmenüeintrag eine Menütrennzeile eingefügt. Das kommerzielle Und-Zeichen **&** dient der Definition von sogenannten Menübeschleunigern in Pulldownmenüs. Das auf das **&**-Zeichen folgende Zeichen definiert dabei eine Taste zur schnellen Selektion des Menüeintrags über die Tastatur. Der Bearbeitungsschlüssel dient der Konfiguration von sogenannten Ghostmenüs. Hierfür ist ein kodierter Integerwert einzutragen, wie er mit den Funktionen **bae_getclassbitfield** oder **bae_getmenubitfield** ermittelt bzw. definiert werden kann (im Zweifelsfall empfiehlt sich die Verwendung des Hexadezimalwertes 80000000h, um sicherzustellen, dass die Funktion immer aktivierbar ist). Der Funktionsrückgabewert ergibt sich zu (-1), wenn ein Fehler aufgetreten ist oder zu Null andernfalls. Mit der Funktion **bae_resetmenuprog** können *sämtliche* Menübelegungen wieder zurückgesetzt werden.

Siehe auch

Funktionen **bae_defmenu**, **bae_defselmenu**, **bae_getclassbitfield**, **bae_getmenubitfield**, **bae_plainmenutext**, **bae_redefmenu**, **bae_resetmenuprog**.

bae_defselmenu - BAE Submenüdefinition starten (STD)

Synopsis

```
int bae_defselmenu(           // Status
      int [0,999];            // Menücode
      int [0,999];            // Menübereichscode:
                              //    1   = Hauptmenü
                              //    101 = erstes Untermenü
                              //    102 = zweites Untermenü
                              //    :   = : Untermenü
      int [0,99];             // Menüstartzeilennummer
      );
```

Beschreibung

Die Funktion **bae_defselmenu** startet die Definition eines Untermenüs im aktuell aktiven BAE-Modul. Der Rückgabewert der Funktion **bae_defselmenu** ist (-1), wenn ein Fehler aufgetreten ist oder Null andernfalls. Im Anschluss an den Aufruf der Funktion **bae_defselmenu** sollten durch wiederholten Aufruf der Funktion **bae_defmenutext** die einzelnen Menüeinträge definiert werden. Die mit **bae_defselmenu** eingeleitete Menüdefinition *muss* durch einen Aufruf der Funktion **bae_endmenu** beendet werden. Mit der Funktion **bae_resetmenuprog** können *sämtliche* Menübelegungen wieder zurückgesetzt werden.

Siehe auch

Funktionen **bae_defmenuprog**, **bae_defmenutext**, **bae_endmenu**, **bae_redefmenu**, **bae_resetmenuprog**.

bae_dialaddcontrol - BAE Dialogelement definieren (STD)

Synopsis

```
int bae_dialaddcontrol(       // Dialogelementindex oder (-1) im Fehlerfall
      int [0,[;               // Parametertyp (STD5)
      int;                    // Minimum int-Parameterwert
      int;                    // Maximum int-Parameterwert
      int;                    // Vorgabe int-Parameterwert
      double;                 // Minimum double-Parameterwert
      double;                 // Maximum double-Parameterwert
      double;                 // Vorgabe double-Parameterwert
      string;                 // Vorgabe string-Parameterwert
      int [0,[;               // Maximallänge string-Parameterwert
      double;                 // Dialogelement X-Position [character-Einheiten]
      double;                 // Dialogelement Y-Position [character-Einheiten]
      double;                 // Dialogelement Dimension [character-Einheiten]
      string;                 // Parameterbezeichnung/Eingabeaufforderung
      );
```

Beschreibung

Die Funktion **bae_dialaddcontrol** definiert ein Dialogelement für den angegebenen Parametertyp. Das Dialogelement wird im nachfolgend mit der Funktion **bae_dialaskparams** aktivierten Dialog an den angegebenen Koordinaten und in der angegebenen Größe angezeigt und mit dem über den Funktionsparameter für die Parameterbezeichnung spezifierten Text nach Bedarf beschriftet. Die Parameterwertvorgaben für das neue Dialogelement sind über die dem angegebenen Parametertyp entsprechenden Funktionsparameter zu übergeben. Das Funktioneergebnis ergibt sich zu (-1) wenn die Generierung des Dialogelements fehlschlägt. Bei erfolgreicher Generierung des Dialogelements wird ein nichtnegativer Dialogelementindex als Funktionsergebnis zurückgeliefert. Dieser Index ist als Selektionsparameter zur Auswahl des Dialogelements bei nachfolgenden Aufrufen der Funktionen **bae_dialgetdata** und **bae_dialsetdata** zu verwenden. Mit **bae_dialaddcontrol** erzeugte Dialogelemente sind bis zum nächsten Aufruf der Funktion **bae_dialclr** gültig bzw. verfügbar.

Siehe auch

Funktionen **bae_dialadvcontrol**, **bae_dialaskparams**, **bae_dialbmpalloc**, **bae_dialboxbufload**, **bae_dialboxbufstore**, **bae_dialclr**, **bae_dialgetdata**, **bae_dialsetdata**.

bae_dialadvcontrol - Erweitertes BAE-Dialogelement setzen (STD)

Synopsis

```
int bae_dialadvcontrol(       // Returns new dialog control index or (-1) on error
      int [0,[;               // Parametertyp (STD5)
      int;                    // Minimum int-Parameterwert
      int;                    // Maximum int-Parameterwert
      int;                    // Vorgabe int-Parameterwert
      double;                 // Minimum double-Parameterwert
      double;                 // Maximum double-Parameterwert
      double;                 // Vorgabe double-Parameterwert
      string;                 // Vorgabe string-Parameterwert
      int [0,[;               // Maximallänge string-Parameterwert
      double;                 // Dialogelement X-Position [character-Einheiten]
      double;                 // Dialogelement Y-Position [character-Einheiten]
      double;                 // Dialogelement Breite [character-Einheiten]
      double;                 // Dialogelement Höhe [character-Einheiten]
      string;                 // Parameterbezeichnung/Eingabeaufforderung
      );
```

Beschreibung

Die Funktion **bae_dialadvcontrol** definiert ein erweitertes Dialogelement (mit Elementhöhenangabe) für den angegebenen Parametertyp. Das Dialogelement wird im nachfolgend mit der Funktion **bae_dialaskparams** aktivierten Dialog an den angegebenen Koordinaten und in der angegebenen Breite und Höhe angezeigt und mit dem über den Funktionsparameter für die Parameterbezeichnung spezifierten Text nach Bedarf beschriftet. Die Parameterwertvorgaben für das neue Dialogelement sind über die dem angegebenen Parametertyp entsprechenden Funktionsparameter zu übergeben. Das Funktioneergebnis ergibt sich zu (-1) wenn die Generierung des Dialogelements fehlschlägt. Bei erfolgreicher Generierung des Dialogelements wird ein nichtnegativer Dialogelementindex als Funktionsergebnis zurückgeliefert. Dieser Index ist als Selektionsparameter zur Auswahl des Dialogelements bei nachfolgenden Aufrufen der Funktionen **bae_dialgetdata** und **bae_dialsetdata** zu verwenden. Mit **bae_dialaddcontrol** erzeugte Dialogelemente sind bis zum nächsten Aufruf der Funktion **bae_dialclr** gültig bzw. verfügbar.

Siehe auch

Funktionen **bae_dialaddcontrol**, **bae_dialaskparams**, **bae_dialbmpalloc**, **bae_dialboxbufload**, **bae_dialboxbufstore**, **bae_dialclr**, **bae_dialgetdata**, **bae_dialsetdata**.

bae_dialaskcall - BAE Dialog mit Listenboxelement-Callbackfunktion aktivieren (STD)

Synopsis

```
int bae_dialaskcall(          // Positiver Aktionscode, oder
                              //    ( 0) bei Betätigung von OK, oder
                              //    (-1) bei Betätigung von Abbruch oder Fehler
      string;                 // Dialogtitel
      int [0,3];              // Ausgabeeinheiten für Distanzabfragen:
                              //    0 = mm
                              //    1 = Inch
                              //    2 = mil
                              //    3 = um
      double ]0.0,[;          // Dialogbreite [character-Einheiten]
      double ]0.0,[;          // Dialoghöhe [character-Einheiten]
      * int;                  // Listboxelement-Callbackfunktion
      );
```

Beschreibung

Die Funktion **bae_dialaskcall** aktiviert einen Dialog mit den über die Funktion **bae_dialaddcontrol** definierten Dialogelementen. In der Titelleiste des Dialogfensters wird der über den ersten Funktionsparameter angegebene Dialogtitel angezeigt. Die Größe des angezeigten Dialogfensters kann über die entsprechenden Funktionsparameter für die Dialogbreite und die Dialoghöhe gesteuert werden. Der Funktionsrückgabewert ergibt sich zu Null wenn die Dialogeingaben über den OK-Button bestätigt wurden. Bei Betätigung einer Schaltlfäche mit einem zugewiesenen positiven Aktionscode gibt **bae_dialaskcall** den entsprechenden Aktionscode zurück. Im Fehlerfall bzw. bei Betätigung des Buttons Abbruch wird der Wert (-1) zurückgegeben. Im Anschluss an einen erfolgreichen Aufruf der Funktion **bae_dialaskcall** können die im Dialog gesetzten Parameterwerte mit der Funktion **bae_dialgetdata** abgefragt werden. Distanz- bzw. Längenangaben werden dabei automatisch entsprechend dem für die Ausgabeeinheiten spezifizierten Parameter der Funktion **bae_dialaskcall** angezeigt bzw. umgerechnet. Der letzte Parameter gestattet die Angabe einer Callbackfunktion die automatisch bei Selektion eines Elements einer im Dialog mit dem Typ **PA_MCALLBACK** definierten Listbox aufgerufen wird.

Listboxelement-Callbackfunktion

```
int callbackfuncname(
      int reason,             // Grund für Callbackfunktionsaufruf
      int boxidx,             // Dialog Index
      int itemidx,            // Listenelement Index
      int itemid,             // Listenelement Id
      string itemstr          // Listenelement Text
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Der Rückgabewert der Callbackfunktion sollte Null sein bei erfolgreicher Abarbeitung. Im Fehlerfall sollte ein Wert ungleich Null zurückgegeben werden.

Siehe auch

Funktionen **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskparams**, **bae_dialbmpalloc**, **bae_dialboxparam**, **bae_dialboxperm**, **bae_dialclr**, **bae_dialgetdata**, **bae_dialsetdata**.

bae_dialaskparams - BAE Dialog aktivieren (STD)

Synopsis

```
int bae_dialaskparams(        // Positiver Aktionscode, oder
                              //    ( 0) bei Betätigung von OK, oder
                              //    (-1) bei Betätigung von Abbruch oder Fehler
                              //    (-2) bei Dialogdimensionsänderung
      string;                 // Dialogtitel
      int [0,3];              // Ausgabeeinheiten für Distanzabfragen:
                              //    0 = mm
                              //    1 = Inch
                              //    2 = mil
                              //    3 = um
      double ]0.0,[;          // Dialogbreite [character-Einheiten]
      double ]0.0,[;          // Dialoghöhe [character-Einheiten]
      );
```

Beschreibung

Die Funktion **bae_dialaskparams** aktiviert einen Dialog mit den über die Funktion **bae_dialaddcontrol** definierten Dialogelementen. In der Titelleiste des Dialogfensters wird der über den ersten Funktionsparameter angegebene Dialogtitel angezeigt. Die Größe des angezeigten Dialogfensters kann über die entsprechenden Funktionsparameter für die Dialogbreite und die Dialoghöhe gesteuert werden. Der Funktionsrückgabewert ergibt sich zu Null wenn die Dialogeingaben über den OK-Button bestätigt wurden. Bei Betätigung einer Schaltlfäche mit einem zugewiesenen positiven Aktionscode gibt **bae_dialaskparams** den entsprechenden Aktionscode zurück. Im Fehlerfall bzw. bei Betätigung des Buttons Abbruch wird der Wert (-1) zurückgegeben. Bei einer Änderung der Dialoggrö&szlige; wird der Wert (-2) zurückgegeben. Im Anschluss an einen erfolgreichen Aufruf der Funktion **bae_dialaskparams** können die im Dialog gesetzten Parameterwerte mit der Funktion **bae_dialgetdata** abgefragt werden. Distanz- bzw. Längenangaben werden dabei automatisch entsprechend dem für die Ausgabeeinheiten spezifizierten Parameter der Funktion **bae_dialaskparams** angezeigt bzw. umgerechnet.

Siehe auch

Funktionen **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskcall**, **bae_dialbmpalloc**, **bae_dialboxperm**, **bae_dialclr**, **bae_dialgetdata**, **bae_dialsetdata**.

bae_dialbmpalloc - BAE Dialogbitmap erzeugen (STD)

Synopsis

```
int bae_dialbmpalloc(         // Nicht-negative Bitmap-Id oder (-1) im Fehlerfall
      double ]0.0,[;          // Gewünschte Bitmapbreite [character-Einheiten]
      double ]0.0,[;          // Gewünschte Bitmaphöhe [character-Einheiten]
      int [2,31];             // Bitmap-Id
      & int;                  // Erzeugte Bitmapbreite [Pixel]
      & int;                  // Erzeugte Bitmaphöhe [Pixel]
      );
```

Beschreibung

Mit der Funktion **bae_dialbmpalloc** können Bitmaps mit den angegebenen Parametern in einer nachfolgend mit **bae_dialaskparams** zu aktivierenden Dialogbox angelegt werden. Der Funktionsrückgabewert ist (-1) bei fehlgeschlagener Bitmapgenerierung oder eine nicht-negative Bitmapidentifikationsnummer bei erfolgreicher Bitmapgenerierung (wobei dann auch die tatsächlich erzeugten Bitmapdimensionen über die entsprechenden Parameter zurückgegeben werden). Nach erfolgreicher Generierung der Bitmap kann diese mit der Funktion **bae_popsetarea** als Grafikausgabeeinheit für die Funktionen **bae_popdrawtext** und **bae_popdrawpoly** selektiert werden.

Siehe auch

Funktionen **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskcall**, **bae_dialaskparams**, **bae_dialboxperm**, **bae_dialclr**, **bae_popdrawpoly**, **bae_popdrawtext**, **bae_popsetarea**.

bae_dialboxbufload - BAE Dialogboxdaten aus Buffer holen (STD)

Synopsis

```
int bae_dialboxbufload(       // Status
      int [1,[;               // Dialogbox Buffer-Id
      );
```

Beschreibung

Die Funktion **bae_dialboxbufload** dient dazu, zuvor mit **bae_dialboxbufstore** gespeicherte Dialogboxdefinition zu laden. Der Funktionsrückgabewert ist Null wenn der Ladevorgang erfolgreich war oder ungleich Null im Fehlerfall.

Siehe auch

Funktionen **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialboxbufstore**, **bae_dialclr**.

bae_dialboxbufstore - BAE Dialogboxdaten in Buffer sichern (STD)

Synopsis

```
int bae_dialboxbufstore(      // Dialogbox Id (>0) oder (-1) im Fehlerfall
      );
```

Beschreibung

Die Funktion **bae_dialboxbufstore** sichert die aktuellen Dialogboxdefinitionen in einem temporären Speicher. Der Funktionsrückgabewert ist ein positiver Integerwert zur Identifikation des benutzten Zwischenspeichers oder (-1) im Fehlerfall. Diese Identifikationsnummer ist in nachfolgenden **bae_dialboxbufload** zum Laden der Dialogboxdaten anzugeben.

Siehe auch

Funktionen **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialboxbufload**, **bae_dialclr**.

bae_dialboxperm - Eigenständigen BAE Dialog aktivieren (STD)

Synopsis

```
int bae_dialboxperm(          // Positive Dialog-Id, oder
                              //    (-1) wenn Dialoganzeige fehlgeschlagen, oder
                              //    (-2) wenn maximale Dialoganzahl erreicht
      string;                 // Dialogtitel
      int [0,3];              // Ausgabeeinheiten für Distanzabfragen:
                              //    0 = mm
                              //    1 = Inch
                              //    2 = mil
                              //    3 = um
      double ]0.0,[;          // Dialogbreite [character-Einheiten]
      double ]0.0,[;          // Dialoghöhe [character-Einheiten]
      );
```

Beschreibung

Die Funktion **bae_dialboxperm** aktiviert einen eigenständigen ("modeless") Dialog mit den über die Funktion **bae_dialaddcontrol** definierten Dialogelementen. In der Titelleiste des Dialogfensters wird der über den ersten Funktionsparameter angegebene Dialogtitel angezeigt. Die Größe des angezeigten Dialogfensters kann über die entsprechenden Funktionsparameter für die Dialogbreite und die Dialoghöhe gesteuert werden. Bei erfolgreicher Generierung der Dialogbox gibt die Funktion die positive Identifikationsnummer für diesen Dialog zurück. Im Fehlerfall wird ein negativer Wert zurückgegeben. Im Anschluss an einen erfolgreichen Aufruf der Funktion **bae_dialboxperm** können (nach Aktivierung dieses Dialogs mit **bae_dialsetcurrent**) die im Dialog gesetzten Parameterwerte mit der Funktion **bae_dialgetdata** abgefragt werden. Distanz- bzw. Längenangaben werden dabei automatisch entsprechend dem für die Ausgabeeinheiten spezifizierten Parameter der Funktion **bae_dialboxperm** angezeigt bzw. umgerechnet.

Siehe auch

Funktionen **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskcall**, **bae_dialaskparams**, **bae_dialbmpalloc**, **bae_dialclr**, **bae_dialgetdata**, **bae_dialsetcurrent**, **bae_dialsetdata**.

bae_dialclr - BAE Dialogelemente löschen (STD)

Synopsis

```
int bae_dialclr(              // Status
      );
```

Beschreibung

Die Funktion **bae_dialclr** löscht alle zuvor mit **bae_dialaddcontrol** erzeugten Dialogelemente. Der Funktionsrückgabewert ist ungleich Null, wenn Dialoge in der aktuellen BAE-Benutzeroberfläche nicht unterstützt werden. Es empfiehlt sich, diese Funktion vor der Neudefinition eines Dialogs auszuführen, um eventuell vorhandene Dialogelemente aus zuvor definierten Dialogen zuverlässig zu löschen. Das Funktionsergebnis sollte ebenfalls geprüft werden, damit bei fehlendem Dialogsupport auf andere interaktive Abfragemechanismen ausgewichen werden kann.

Siehe auch

Funktionen **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskcall**, **bae_dialaskparams**, **bae_dialbmpalloc**, **bae_dialboxbufload**, **bae_dialboxbufstore**, **bae_dialboxperm**, **bae_dialgetdata**, **bae_dialsetdata**.

bae_dialgetdata - BAE Dialogelementdaten abfragen (STD)

Synopsis

```
int bae_dialgetdata(          // Status
      int [0,[;               // Dialogelementindex
      & int;                  // Dialogelement integer-Wert
      & double;               // Dialogelement double-Wert
      & string;               // Dialogelement string-Wert
      );
```

Beschreibung

Mit der Funktion **bae_dialgetdata** können nach einem erfolgreichen Aufruf der Funktion **bae_dialaskparams**, die vom Anwender gesetzten Dialogparameterwerte abgefragt werden. Die Auswahl des abzufragenden Dialogelements erfolgt durch Spezifikation des von **bae_dialaddcontrol**, gelieferten Dialogelementindex. Die Rückgabe des ermittelten Parameterwertes erfolgt über den Funktionsparameter, der dem Datentyp des abgefragten Dialogelements entspricht. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterabfrage bzw. ungleich Null bei fehlgeschlagener Abfrage.

Siehe auch

Funktionen **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskparams**, **bae_dialclr**, **bae_dialsetdata**.

bae_dialgettextlen - BAE Dialogtextlänge abfragen (STD)

Synopsis

```
double bae_dialgettextlen(    // Rückgabe Textlänge [character-Einheiten]
      int [0,[;               // Dialogtexttyp bzw. Zeichensatztyp
      string;                 // Dialogtextzeichenkette
      );
```

Beschreibung
Die Funktion **bae_dialgettextlen** ermittelt den Platzbedarf für den angegebenen Dialogtext.

bae_dialsetcurrent - Aktuelle BAE-Dialogbox setzen (STD)

Synopsis

```
int bae_dialsetcurrent(         // Status
      int [0,[;                 // Dialogbox-Id
      );
```

Beschreibung

Die Funktion **bae_dialsetcurrent** aktiviert die über die Identifikationsnummer angebebene (eigenständige) Dialogbox für nachfolgende Dialogboxfunktionen. Bei erfolgreicher Dialogauswahl ergibt sich der Funktionsrückgabewert zu Null. Bei fehlgeschlagener Dialogauswahl wird ein Wert ungleich Null zurückgegeben.#

Siehe auch

Funktionen **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskcall**, **bae_dialaskparams**, **bae_dialbmpalloc**, **bae_dialboxperm**, **bae_dialclr**, **bae_dialgetdata**, **bae_dialsetdata**.

bae_dialsetdata - BAE Dialogelementdaten setzen (STD)

Synopsis

```
int bae_dialsetdata(            // Status
      int [0,[;                 // Dialogelement Index
      int [0,[;                 // Dialogelement Parametertyp (STD5)
      int;                      // Dialogelement integer-Wert
      double;                   // Dialogelement double-Wert
      string;                   // Dialogelement string-Wert
      );
```

Beschreibung

Die Funktion **bae_dialsetdata** dient dazu, Parametertypen bzw. Parameterwerte für zuvor mit **bae_dialaddcontrol** definierte Dialogelemente zu setzen. Die Auswahl des zu ändernden Dialogelements erfolgt durch Spezifikation des von **bae_dialaddcontrol**, gelieferten Dialogelementindex. Der neue Parameterwert ist über den dem angegebenen Parametertyp entsprechenden Funktionsparameter zu übergeben. Der Funktionsrückgabewert ist Null bei erfolgreicher Änderung des Dialogelements bzw. ungleich Null im Fehlerfall.

Siehe auch

Funktionen **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskparams**, **bae_dialclr**, **bae_dialgetdata**.

bae_endmainmenu - BAE Hauptmenüdefinition beenden (STD)

Synopsis

```
int bae_endmainmenu(            // Status
      );
```

Beschreibung

Die Funktion **bae_endmainmenu** beendet die im aktuell aktiven BAE-Modul mit der Funktion **bae_redefmainmenu** eingeleitete Neudefinition des Hauptmenüs. Der Rückgabewert der Funktion **bae_endmainmenu** ist (-1), wenn ein Fehler aufgetreten ist oder Null andernfalls. Mit der Funktion **bae_resetmenuprog** können *sämtliche* Menübelegungen wieder zurückgesetzt werden.

Siehe auch

Funktionen **bae_redefmainmenu**, **bae_resetmenuprog**.

bae_endmenu - BAE Menüdefinition beenden (STD)

Synopsis

```
int bae_endmenu(                // Status
      );
```

Beschreibung

Die Funktion **bae_endmenu** beendet die im aktuellen BAE-Modul mit einer der Funktionen **bae_defmenu** oder **bae_defselmenu** eingeleitete Menüdefinition. Der Rückgabewert der Funktion **bae_endmenu** ist (-1), wenn ein Fehler aufgetreten ist oder Null andernfalls. Mit der Funktion **bae_resetmenuprog** können *sämtliche* Menübelegungen wieder zurückgesetzt werden.

Siehe auch

Funktionen **bae_defmenu**, **bae_defselmenu**, **bae_resetmenuprog**.

bae_fontcharcnt - BAE Anzahl Zeichensatzzeichen (STD)

Synopsis

```
int bae_fontcharcnt(            // Zeichenanzahl
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_fontcharcnt** entspricht der Anzahl der im aktuell geladenen Zeichensatz definierten Zeichen.

bae_fontname - BAE Zeichensatzname abfragen (STD)

Synopsis

```
string bae_fontname(            // Zeichensatzname
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_fontname** entspricht dem Namen des für das aktuell geladene **AutoEngineer** Planelement gültigen Zeichensatzes.

bae_getactmenu - BAE aktives Menü abfragen (STD)

Synopsis

```
int bae_getactmenu(             // Menüfunktion (STD4)
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_getactmenu** entspricht der im **AutoEngineer** aktuell aktiven Menüfunktion (STD4) oder (-1) wenn keine Menüfunktion abgearbeitet wird. Diese Funktion wird benötigt für Programme, die auf Tasten gelegt werden.

bae_getanglelock - BAE Winkelfreigabeflag abfragen (STD)

Synopsis

```
int bae_getanglelock(           // Winkelfreigabe Flag (STD9)
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_getanglelock** entspricht dem im **AutoEngineer** aktuell eingestellten Wert des Winkelfreigabeflags (0=Winkel freigeben, 1=Winkel einhalten).

Siehe auch

Funktion **bae_setanglelock**.

bae_getbackgrid - BAE Hintergrundraster abfragen (STD)

Synopsis

```
void bae_getbackgrid(
      & double;              // X-Hintergrundraster (STD2)
      & double;              // Y-Hintergrundraster (STD2)
      );
```

Beschreibung

Die Funktion **bae_getbackgrid** gibt in den beiden Parametern die Werte des aktuell im **AutoEngineer** eingestellten Hintergrundrasters für X- und Y-Richtung zurück. Werte von Null spezifizieren, dass kein Hintergrundraster angezeigt wird.

Siehe auch

Funktion **bae_setbackgrid**.

bae_getcasstime - Zeitpunkt des letzten Projektnetzlistenupdates durch Packager/Backannotation ermitteln (STD)

Synopsis

```
string bae_getcasstime(       // Rückgabe Packager-Netzlistenname
      & int;                 // Rückgabe Zeitangabe Sekunde
      & int;                 // Rückgabe Zeitangabe Minute
      & int;                 // Rückgabe Zeitangabe Stunde
      & int;                 // Rückgabe Datumsangabe Tag
      & int;                 // Rückgabe Datumsangabe Monat
      & int;                 // Rückgabe Datumsangabe Jahr
      );
```

Beschreibung

Mit der Funktion **bae_getcasstime** kann der Zeitpunkt des zuletzt durch **Packager** bzw. Backannotation erfolgten Projektnetzlistenupdates ermittelt werden.

Siehe auch

Funktionen **bae_getpackdata**, **bae_getpacktime**.

bae_getclassbitfield - BAE DDB-Klasse Bearbeitungsschlüssel abfragen (STD)

Synopsis

```
int bae_getclassbitfield(     // Rückgabe Bearbeitungsschlüssel
      int ]0,[;              // DDB-Datenbankklasse (STD1)
      );
```

Beschreibung

Die Funktion **bae_getclassbitfield** ermittelt den Bearbeitungsschlüssel, der der spezifizierten DDB-Datenbankklasse zugewiesen ist. Der Rückgabewert der Funktion **bae_getclassbitfield** ist (-1) bei Spezifikation einer ungültigen bzw. unbekannten Datenbankklasse.

Der Bearbeitungsschlüssel ist ein kodierter Integerwert, der innerhalb eines BAE-Moduls für jede bearbeitbare DDB-Klasse eindeutig ist. Durch bitweises Verodern dieser Kodierungen können weitere Bearbeitungsschlüssel generiert und anschließend mit einer der Funktionen **bae_defmenutext**, **bae_defmenuprog** oder **bae_redefmenu** selektiv an Menüeinträge zugewiesen werden. Damit können Menüfunktionen so konfiguriert werden, dass sie nur auf bestimmte Datenbankklassen anwendbar sind.

Siehe auch
Funktionen **bae_defmenuprog**, **bae_defmenutext**, **bae_getmenubitfield**, **bae_redefmenu**.

bae_getcmdbuf - BAE Kommandohistorie abfragen (STD)

Synopsis

```
int bae_getcmdbuf(              // Status
      int [-50,2099];           // Kommandospeicherindex 0 bis 49
                                //   oder 1000 bis 1099 für Undo-Einträge 1 bis 100
                                //   oder 2000 bis 2099 für Redo-Einträge 1 bis 100
                                //   oder negativer Wert für Meldungshistorie
      & string;                 // Kommando- bzw. Kommandosequenzzeichenkette
      & string;                 // Kommandoanzeigetext
      );
```

Beschreibung

Die Funktion **bae_getcmdbuf** dient dazu, die Kommandohistorie für das aktuelle Kontextmenü abzufragen. Dies ist die Liste der zuletzt über die rechte Maustaste aktivierten Kommandos. Der Kommandospeicherindex gibt die Position des abzufragenden Kommandos an (Null ist das zuletzt ausgeführte Kommando). Die Kommando- bzw. Kommandosequenzzeichenkette und der Kommandoanzeigetext (entsprechend der Titelleistenanzeige) werden über den zweiten und dritten Funktionsparameter an den Aufrufer zurückgegeben. Der Funktionsrückgabewert ist 1 wenn die Abfrage erfolgreich war oder Null wenn ein ungültiger Kommandospeicherindex angegeben wurde.

Siehe auch

Funktion **bae_storecmdbuf**.

bae_getcolor - BAE Farbwert abfragen (STD)

Synopsis

```
int bae_getcolor(               // Farbwert (STD18)
      int;                      // Anzeigeelementtyp (SCM1|LAY9|ICD9)
      );
```

Beschreibung

Die Funktion **bae_getcolor** ermittelt den Farbwert für den angegebenen Anzeigeelementtyp. Der Wert für den Anzeigeelementtyp muss entsprechend der aktuellen Interpreterumgebung gesetzt sein.

Siehe auch

Funktion **bae_setcolor**.

bae_getcoorddisp - BAE Koordinatenanzeige abfragen (STD)

Synopsis

```
int bae_getcoorddisp(           // Koordinatenanzeigemodus (STD7)
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_getcoorddisp** entspricht dem im AutoEngineer aktuell eingestellten Koordinatenanzeigemodus, wobei der Wert 0 für mm-Einheiten (bzw. Mikrometer-Einheiten im **IC-Design** System) und der Wert 1 für Inch-Einheiten (bzw. mil-Einheiten im **IC-Design** System) steht.

Siehe auch

Funktion **bae_setcoorddisp**.

bae_getdblpar - BAE Doubleparameter abfragen (STD)

Synopsis

```
int bae_getdblpar(             // Status
      int [0,[;                // Parametertyp/-nummer:
                               //    0 = maximale Dialogboxbreite
                               //    1 = maximale Dialogboxhöhe
                               //    2 = Grafikanzeige Zoomfaktor
                               //    3 = Gummibandeckradius (STD2)
                               //    4 = Gummiband X-Vektorkoordinate (STD2)
                               //    5 = Gummiband Y-Vektorkoordinate (STD2)
                               //    6 = Fixierte X-Pickkoordinate (STD2)
                               //    7 = Fixierte Y-Pickkoordinate (STD2)
                               //    8 = Dialogboxbreite
                               //    9 = Dialogboxhöhe
                               //    10 = Bildschirmpickbereich (STD2)
                               //    11 = Elementauswahl relativer
                               //           Vorschaubereich [0.05, 0.95]
                               //    12 = Dialogbox X-Einheiten Pixel
                               //    13 = Dialogbox Y-Einheiten Pixel
      & double;                // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **bae_getdblpar** dient der Abfrage von im **Bartels AutoEngineer** gesetzten Parametern vom Typ `double`. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **bae_getintpar**, **bae_getstrpar**, **bae_setdblpar**, **bae_setintpar**, **bae_setstrpar**.

bae_getfuncprog - BAE Funktionstastenprogrammierung abfragen (STD)

Synopsis

```
string bae_getfuncprog(        // Programmname
      int [1,128];             // Funktionstastennummer
      );
```

Beschreibung

Die Funktion **bae_getfuncprog** ermittelt den Namen des **User Language**-Programms, welches der angegebenen Standardtaste zugewiesen ist. Ist der Taste eine BAE-Menüfunktion zugewiesen, dann wird die entsprechende BAE-Menüfunktionsnummer mit einem vorangestellten Hashzeichen (#) zurückgegeben. Der Funktionsrückgabewert ist ein Leerstring, wenn keine Zuweisung für die Funktionstaste definiert ist bzw. wenn eine ungültige Funktionstastennummer spezifiziert wurde.

Siehe auch

Funktionen **bae_deffuncprog**, **bae_resetmenuprog**.

bae_getgridlock - BAE Rasterfreigabeflag abfragen (STD)

Synopsis

```
int bae_getgridlock(           // Rasterfreigabe Flag (STD8)
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_getgridlock** entspricht dem im **AutoEngineer** aktuell eingestellten Wert des Rasterfreigabeflags (0=Raster freigeben, 1=Raster einhalten).

Siehe auch

Funktion **bae_setgridlock**.

bae_getgridmode - BAE Rasterabhängigkeitsmodus abfragen (STD)

Synopsis

```
int bae_getgridmode(          // Rückgabe Modus für automatische
                              //  Rastereinstellung:
                              //    0x01: Eingaberaster = 0.25 × Hintergrundraster
                              //    0x02: Eingaberaster = 0.50 × Hintergrundraster
                              //    0x04: Eingaberaster = 1.00 × Hintergrundraster
                              //    0x08: Eingaberaster = 2.00 × Hintergrundraster
                              //    0x10: Hintergrundraster = 0.25 × Eingaberaster
                              //    0x20: Hintergrundraster = 0.50 × Eingaberaster
                              //    0x40: Hintergrundraster = 1.00 × Eingaberaster
                              //    0x80: Hintergrundraster = 2.00 × Eingaberaster
      );
```

Beschreibung

Die Funktion **bae_getgridmode** dient der Abfrage des aktuell aktiven BAE Rasterabhängigkeitsmodus.

Siehe auch

Funktion **bae_setgridmode**.

bae_getinpgrid - BAE Eingaberaster abfragen (STD)

Synopsis

```
void bae_getinpgrid(
      & double;               // X-Eingaberaster (STD2)
      & double;               // Y-Eingaberaster (STD2)
      );
```

Beschreibung

Die Funktion **bae_getinpgrid** gibt in den beiden Parametern die Werte des aktuell im **AutoEngineer** eingestellten Eingaberasters für X- und Y-Richtung zurück.

Siehe auch

Funktion **bae_setinpgrid**.

bae_getintpar - BAE Integerparameter abfragen (STD)

Synopsis

```
int bae_getintpar(            // Status
      int [0,[;               // Parametertyp/-nummer:
                              //    0 = Koordinateneingaben Bereichsprüfung:
                              //        0 = Bereichsprüfung aktiviert
                              //        1 = Bereichsprüfung deaktiviert
                              //    1 = Modulwechsel Autosavemodus:
                              //        0 = Autosave ohne Benutzerabfrage
                              //        1 = Benutzerabfrage vor Autosave
                              //    2 = Anzeigedeaktivierungsmodus:
                              //        0 = Anzeige aktiviert
                              //        1 = Anzeige deaktiviert
                              //    3 = Benutzeroberfläche Menü-/Mausmodus:
                              //        0 = Seitenmenü-Konfiguration
                              //        1 = Pulldownmenü, LMB-Kontext
                              //        2 = Pulldwonmenü, RMB-Kontext
                              //    4 = Arbeitsbereichstext Farbauswahl:
                              //        0 = Standardfarben
                              //        1 = invertierte Standardfarben
                              //        2 = arbeitsbereichsspezifische Farben
                              //    5 = Element Laden Anzeigemodus:
                              //        0 = Übersichtsanzeige nach Laden
                              //        1 = Anzeige aktiviert durch bae_load
                              //    6 = Dateiauswahl Dialogmodus:
                              //        0 = alte BAE-Dateiauswahl
                              //        1 = Explorer Standardansicht
                              //        2 = Explorer Listenansicht
                              //        3 = Explorer Detailansicht
                              //        4 = Exploreransicht kleine Piktogramme
                              //        5 = Exploreransicht große Piktogramme
                              //        6 = Standard-Stil und
                              //             Standard-Größe benutzen
                              //    7 = Elementauswahl Dialogmodus:
                              //        0 = nur Name anzeigen
                              //        1 = Name und Datum anzeigen
                              //    8 = Elementauswahl Sortierfunktion:
                              //        0|1 = Sortierung nach Name
                              //        2 = Sortierung numerisch
                              //        3 = Sortierung nach Datum
                              //    9 = Anzeigemodus Platzierung:
                              //        0 = platzierte Elemente sichtbar
                              //        1 = unplatzierte Elemente sichtbar
                              //    10 = Letzter Dateisystemfehler
                              //    11 = Kommandohistorie Modus:
                              //        0 = Kommandohistorie aktiviert
                              //        1 = Kommandohistorie deaktiviert
                              //    12 = Popupmenü Mauswarpmodus:
                              //        0 = Kein Popupmenü Mauswarp
                              //        1 = Popupmenü Mauswarp erstes Element
                              //        2 = Popupmenü Mauswarp
                              //             vorselektiertes Element
                              //        +4 = Mauspositionsrestore-Warp
                              //        +8 = Elementpickpositions-Warp
                              //    13 = Sicherungsmodus:
                              //        0 = Sicherung aktiviert
                              //        1 = Sicherung deaktiviert
                              //    14 = Minimalgröße Mausrechteck:
                              //        ]0,[ = minimale Mausrechteckgröße
                              //    15 = Mausinfo-Anzeigemodus:
                              //        0 = Tooltip-Infoanzeige
                              //        1 = Kontinuierliche Fadenkreuz-Infoanzeige
                              //        2 = Fadenkreuz-Infoanzeige mit Strg
                              //    16 = Nächste Dialogbox-Identifikationsnummer
                              //    17 = Zuletzt erzeugte
                              //          Tooltipidentifikationsnummer
```

```
                              //    18 = Polygonverwurfsanzahl
                              //    19 = Polygonprüfungsabschaltung:
                              //        0 = Polygonprüfung aktiv
                              //        1 = Polygonprüfung abgeschaltet
                              //    20 = Kursortastenrastermodus:
                              //        0 = Eingaberaster
                              //        1 = Pixelraster
                              //    21 = Flag - Element nicht gesichert
                              //    22 = Elementbatchlademodus:
                              //        0 = keinen Batch laden
                              //        1 = Batch laden
                              //        2 = Batch laden, Zoomfenster restaurieren
                              //    23 = Rasterlinienanzeige:
                              //        0 = Punktraster
                              //        1 = Linienraster
                              //    24 = Flag - Spiegelungsanzeige
                              //    25 = Flag - Eingaberasteranzeige
                              //    26 = Mausfunktionswiederholungsmodus:
                              //        0 = Menüfunktion wiederholen
                              //        +1 = Tastenfunktion wiederholen
                              //        +2 = Kontextmenüfunktion wiederholen
                              //    27 = Maximale Undo-Redo-Anzahl
                              //    28 = Menübaumansichtsmodus:
                              //        0 = Kein Menübaumfenster
                              //        1 = Menübaumfenster links
                              //        2 = Menübaumfenster rechts
                              //    29 = Menübaumansicht Pixelbreite
                              //    30 = Flag - Meldungshistorie deaktiviert
                              //    31 = Elementlademeldungsmodus:
                              //        0 = Standardmeldung
                              //        1 = Benutzerspezifische Meldung
                              //        2 = Benutzerspezifische Fehlermeldung
                              //    32 = Pickmarkeranzeigemodus:
                              //        0 = Kreismarker
                              //        1 = Diamantmarker
                              //    33 = Mausdrag-Status:
                              //        0 = Kein Mausdrag
                              //        1 = Mausdragangeforderung
                              //        2 = Mausdrag durchgeführt
                              //        3 = Mausdragfreigabeanforderung
                              //    34 = Flag - Menüfunktionswiederholung
                              //          angefordert
                              //    35 = Flag - Funktion abgebrochen
                              //    36 = Flag - Planauswahlvorschau
                              //    37 = Dateizugriffsfehler-Anzeigemodus:
                              //        0 = Nur Statuszeilenanzeige
                              //        1 = Bestätigungsabfrage
                              //    38 = Elementauswahl-Referenzanzeigemodus:
                              //        0 = Projektdatei-Referenzen anzeigen
                              //        1 = Bibliotheksdatei-Referenzen anzeigen
                              //    39 = Maus-Doppelklick-Modus:
                              //        0 = Doppelklick und Selektion
                              //             von 0 an rechte Maustaste zuweisen
                              //        1 = Doppelklick ignorieren
                              //        2 = Doppelklick an rechte Maustaste
                              //             zuweisen
                              //    40 = Mauspick-Doppelklick-Modus:
                              //        0 = Doppelklick an rechte Maustaste
                              //             zuweisen
                              //        1 = Doppelklick ignorieren
                              //    41 = Flags - Dialogelementunterstützung:
                              //        0 = Keine Dialogelemente unterstützt
                              //        |1 = Dialogbasiselemente unterstützt
                              //        |2 = Listenanzeige unterstützt
                              //        |4 = Fortschittsanzeige unterstützt
                              //        |8 = Werkzeugleistenschaltfläche
                              //              unterstützt
```

```
                              //    42 = Fortschrittsanzeigemodus:
                              //        0 = Keine Fortschrittsanzeige
                              //        1 = Fortschrittsanzeige
                              //    43 = Abbruchanforderungsflag für
                              //          Fortschrittsanzeige
                              //    44 = Flag - Mittlere Maustaste deaktiviert
                              //    45 = Anzahl aktueller Undo-Elemente
                              //    46 = Flag - Datei Drag-und-Drop Operation
                              //    47 = Flag - Automatische
                              //          BAE-Fensteraktivierung
                              //    48 = Anzahl aktive Menüfunktionen
                              //    49 = Anzahl Punkte in interner Polygonliste
                              //    50 = Priorit&auuml;t alternative
                              //          Konfigurationsdatei
                              //    51 = Sicherungsmodus für Dialogposition:
                              //        0 = Absolutkoordinaten specihern
                              //        1 = Hauptfenster-Relativkoordinaten
                              //             speichern
                              //        2 = Hauptfenster-Monitorabsolutkoordinaten
                              //             speichern
                              //    52 = Message-Box Default-Button-Index:
                              //        (-1) = Kein Default (Abbruch oder No
                              //        0-2 = Default-Button-Index
      & int;                  // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **bae_getintpar** dient der Abfrage von im **Bartels AutoEngineer** gesetzten Integerparametern. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **bae_getdblpar**, **bae_getstrpar**, **bae_setdblpar**, **bae_setintpar**, **bae_setstrpar**.

bae_getinvcolor - BAE Farbinvertierungsmodus abfragen (STD)

Synopsis

```
int bae_getinvcolor(          // Rückgabe Farbeinvertierungsmodus
      );
```

Beschreibung

Die Funktion **bae_getinvcolor** überprüft, ob die BAE-Systemfarbpalette "invertiert" ist. Die Systemfarbpalette wird als invertiert betrachtet, wenn die dunkelste Farbe in der aktuell (über die Datei `bae.col` im BAE-Programmverzeichnis) definierten Systemfarbpalette nicht schwarz ist. Die Funktion gibt einen Wert ungleich Null zurück, wenn die BAE-Systemfarbpalette invertiert ist; andernfalls ist der Funktionsrückgabewert Null.

bae_getmenubitfield - BAE Menüfunktion Bearbeitungschlüssel abfragen (STD)

Synopsis

```
int bae_getmenubitfield(      // Rückgabe Bearbeitungsschlüssel
      int [0,999];            // Menücode
      int [0,99];             // Menüzeile
      );
```

Beschreibung

Die Funktion **bae_getmenubitfield** ermittelt den Bearbeitungsschlüssel, der dem angegebenen Menüeintrag zugewiesen ist. Die Spezifikation des Menüeintrags erfolgt durch die Angabe der Hauptmenünummer über den Parameter für den Menücode sowie durch die Angabe der Menüzeile im zugehörigen Untermenü. Der Rückgabewert der Funktion **bae_getmenubitfield** ist (-1), wenn die Spezifikation des Menüeintrags ig ist.

Der Bearbeitungsschlüssel ist ein kodierter Integerwert, der angibt, für welche aktuell geladenen Elemente die Menüfunktion aufgerufen werden kann. Mit Hilfe dieses Schlüssels können Menüeinträge im **Bartels AutoEngineer** in Abhängigkeit vom Typ des aktuell geladenen Elements wahlweise aktiviert bzw. deaktiviert werden. Dieser Mechanismus kommt insbesondere bei der Konfiguration der sogenannten Ghostmenüs in den Windows-Versionen der BAE-Software zur Anwendung. Ist der Bearbeitungschlüssel mit dem Hexadezimalwert 80000000h kodiert, dann kann die entsprechende Funktion *immer* (d.h. auch wenn kein Element geladen ist) aktiviert werden. Der Hexadezimalwert 7FFFFFFFh gibt an, dass die Funktion für jeden beliebigen geladenen Elementtyp aufgerufen werden kann. Weitere Kodierungen sind in den einzelnen BAE-Modulen in Abhängigkeit von den bearbeitbaren DDB-Datenbankklassen definiert. Die Kodierung für jede einzelne Datenbankklasse kann mit der Funktion **bae_getclassbitfield** ermittelt werden. Durch bitweises Verodern dieser Kodierungen können weitere Bearbeitungsschlüssel generiert und anschließend mit einer der Funktionen **bae_defmenutext**, **bae_defmenuprog** oder **bae_redefmenu** selektiv an Menüeinträge zugewiesen werden. Damit ist die Konfiguration benutzerdefinierter Ghostmenüs möglich.

Siehe auch

Funktionen **bae_defmenuprog**, **bae_defmenutext**, **bae_getclassbitfield**, **bae_redefmenu**.

bae_getkeyprog - BAE Standardtastenprogrammierung abfragen (STD)

Synopsis

```
string bae_getkeyprog(        // Programmname
      int;                    // Tastenzeichen
      );
```

Beschreibung

Die Funktion **bae_getkeyprog** ermittelt den Namen des **User Language**-Programms, welches der angegebenen Standardtaste zugewiesen ist. Ist der Taste eine BAE-Menüfunktion zugewiesen, dann wird die entsprechende BAE-Menüfunktionsnummer mit einem vorangestellten Hashzeichen (**#**) zurückgegeben. Der Funktionsrückgabewert ist ein Leerstring, wenn keine Zuweisung für die Standardtaste definiert ist bzw. wenn ein iges Tastenzeichen spezifiziert wurde.

Siehe auch

Funktionen **bae_defkeyprog**, **bae_resetmenuprog**.

bae_getmenuitem - BAE Menüeintrag abfragen (STD)

Synopsis

```
int bae_getmenuitem(            // Status
      & int;                    // Menücode
      & string;                 // Menütext(e)
      & string;                 // Menükommando
      );
```

Beschreibung

Die Funktion **bae_getmenuitem** ermöglicht die Anwahl einer BAE-Menüfunktion zur Abfrage der Eigenschaften des selektierten Menüeintrags. Im Menücodeparameter wird der Aufrufcode für den selektierten Menüeintrag zurückgeliefert. Im Parameter für den Menütext werden getrennt durch **->** die zur Selektion des Menüeintrags angewählten Menütext(e) der hierarchischen Menüstruktur zurückgeliefert. Im Parameter für das Menükommando wird die dem Menüeintrag zugewiesene Menükommandosequenz zurückgeliefert. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage, oder ungleich Null bei fehlgeschlagener bzw. abgebrochener Abfrage.

Siehe auch

Funktionen **bae_callmenu**, **bae_defmenuprog**, **bae_getmenuprog**, **bae_getmenutext**.

bae_getmenuprog - BAE Menüprogrammierung abfragen (STD)

Synopsis

```
string bae_getmenuprog(         // Programmname
      int [0,999];              // Menücode
      int [0,99];               // Menüzeile
      );
```

Beschreibung

Die Funktion **bae_getmenuprog** ermittelt den Namen des **User Language**-Programms, welches dem angegebenen Menüeintrag zugewiesen ist. Ist dem Menüeintrag eine BAE-Menüfunktion zugewiesen, dann wird die entsprechende BAE-Menüfunktionsnummer mit einem vorangestellten Hashzeichen (**#**) zurückgegeben. Der Menücode gibt die Nummer des Hauptmenüs an, die Menüzeile die Position im zugehörigen Submenü. Der Funktionsrückgabewert ist ein Leerstring, wenn keine Zuweisung für den Menüeintrag definiert ist bzw. wenn die Spezifikation des Menüeintrags ig ist.

Siehe auch

Funktionen **bae_defmenuprog**, **bae_getmenuitem**, **bae_getmenutext**, **bae_resetmenuprog**.

bae_getmenutext - BAE Menütext abfragen (STD)

Synopsis

```
string bae_getmenutext(         // Menütext
      int [0,999];              // Menücode
      int [0,99];               // Menüzeile
      );
```

Beschreibung

Die Funktion **bae_getmenutext** ermittelt den Anzeigetext, der dem angegebenen Menüeintrag zugewiesen ist. Der Menücode gibt die Nummer des Hauptmenüs an, die Menüzeile die Position im zugehörigen Submenü. Der Funktionsrückgabewert ist ein Leerstring, wenn die Spezifikation des Menüeintrags ig ist bzw. der spezifizierte Menüeintrag nicht existiert.

Siehe auch

Funktionen **bae_defmenuprog**, **bae_getmenuitem**, **bae_getmenuprog**, **bae_plainmenutext**, **bae_resetmenuprog**.

bae_getmoduleid - BAE Modulbezeichnung abfragen (STD)

Synopsis

```
string bae_getmoduleid(          // Modulbezeichnung
      );
```

Beschreibung

Die Funktion **bae_getmoduleid** ermittelt die (mit **bae_setmoduleid** gesetzte) Bezeichnung des aktuell aktiven BAE-Programmmoduls.

Siehe auch

Funktion **bae_setmoduleid**.

bae_getmsg - BAE HighEnd Message empfangen (STD/HighEnd)

Synopsis

```
string bae_getmsg(               // Rückgabe Messagestring
      );
```

Beschreibung

Die Funktion **bae_getmsg** ist nur in **BAE HighEnd** verfügbar. Mit **bae_getmsg** können Nachrichten empfangen werden, die zuvor mit der Funktion **bae_sendmsg** an die anderen Module eine **BAE HighEnd**-Sitzung abgesendet wurden. Zu einer Sitzung gehören alle Programminstanzen, die ausgehend von einem BAE-Aufruf über die Funktion Weiteres Fenster aus dem BAE-Hauptmenü oder dem **Schaltplaneditor** gestartet wurden. In den addressierten Modulen wird bei Ankunft einer Nachricht automatisch das **User Language**-Programm bae_msg gestartet. Ist **bae_msg** nicht verfügbar, dann erfolgt der Aufruf eines **User Language**-Programms mit modulspezifischem Namen (**scm_msg** im **Schaltplaneditor**, **ged_msg** im **Layouteditor**, **ar_msg** im **Autorouter**, etc.). Das automatisch gestartete **User Language**-Programm muss die Nachrichtmit der Funktion **bae_getmsg** entgegennehmen. Die Nachricht steht nur während der Dauer dieses Programmaufrufs zur Verfügung. Wird die Nachricht von dem ***_msg**-Programm nicht angenommen, dann geht sie verloren. Der Inhalt der Nachricht kann dazu benutzt werden, eine spezielle Aktion im empfangenden Modul auszulösen.

Siehe auch

Funktion **bae_sendmsg**.

bae_getpackdata - Daten des letzten Projekt-Packagerlaufs abfragen (STD)

Synopsis

```
int bae_getpackdata(             // Status
      string;                    // Projektdateiname
      & string;                  // Rückgabe Layoutbibliotheksdatei
      & string;                  // Rückgabe Netzlistenname
      );
```

Beschreibung

Die Funktion **bae_getpackdata** ermittelt die Parameter Layoutbibliotheksdateiname und Netzlistename des für die angebene Projektdatei zuletzt durchgeführten **Packager**-Laufs. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder ungleich Null wenn die Parameterdaten nicht gefunden werden konnten.

Siehe auch

Funktionen **bae_getcasstime**, **bae_getpacktime**.

bae_getpacktime - Datum/Uhrzeit des letzten Projekt-Packagerlaufs abfragen (STD)

Synopsis

```
int bae_getpacktime(           // Status
      string;                  // Projektdateiname
      & int;                   // Rückgabe Zeitangabe Sekunde
      & int;                   // Rückgabe Zeitangabe Minute
      & int;                   // Rückgabe Zeitangabe Stunde
      & int;                   // Rückgabe Datumsangabe Tag
      & int;                   // Rückgabe Datumsangabe Monat
      & int;                   // Rückgabe Datumsangabe Jahr
      );
```

Beschreibung

Die Funktion **bae_getpacktime** ermittelt das Datum und die Uhrzeit des für die angebene Projektdatei zuletzt durchgeführten **Packager**-Laufs. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder ungleich Null wenn die Zeitangaben nicht ermittelt werden konnten.

Siehe auch

Funktionen **bae_getcasstime**, **bae_getpackdata**.

bae_getpolyrange - Bereich der internen BAE-Polygonpunktliste abfragen (STD)

Synopsis

```
int bae_getpolyrange(          // Status
      & double;                // Rückgabe linke Polygongrenze
      & double;                // Rückgabe untere Polygongrenze
      & double;                // Rückgabe rechte Polygongrenze
      & double;                // Rückgabe obere Polygongrenze
      );
```

Beschreibung

Die Funktion **bae_getpolyrange** dient der Abfrage der Ausdehnung bzw. Begrenzung des mit **bae_storepoint** definierten internen Polypons. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder ungleich Null im Fehlerfall.

Siehe auch

Funktionen **bae_clearpoints**, **bae_storepoint**.

bae_getstrpar - BAE Stringparameter abfragen (STD)

Synopsis

```
int bae_getstrpar(            // Status
      int [0,[;               // Parametertyp/-nummer:
                              //    0 = Aktueller Element-Kommentartext
                              //    1 = Aktuelle Element-Spezifikation
                              //    2 = Letzte Dateizugriffsfehler-Datei
                              //    3 = Letztes Dateizugriffsfehler-Element
                              //  [ 4 = Systemparameter - kein Lesezugriff ]
                              //  [ 5 = Systemparameter - kein Lesezugriff ]
                              //    6 = zuletzt geladene Farbtabelle
                              //    7 = Menütext der zuletzt aufgerufenen Funktion
                              //    8 = Aktueller Menüitemelementtext
                              //    9 = Aktuelle benutzerspezifische
                              //         Elementlademeldung
                              //    10 = Zwischenablage-Textstring
                              //    11 = Modulaufruf nächstes Dateiargument
                              //    12 = Modulaufruf nächstes Elementargument
                              //    13 = Modulaufruf nächstes
                              //          Kommando-/Typargument
                              //    14 = Letzter Ausgabedateiname
                              //    15 = Hostname
                              //  [ 16 = Systemparameter - kein Lesezugriff ]
                              //    17 = Datenverzeichnis für alle Benutzer
                              //    18 = Datenverzeichnis für aktuellen Benutzer
                              //    19 = Alternatives Verzeichnis für
                              //          Konfigrationsdaten
                              //    20 = Spalte lokale Daten
                              //    21 = Spalt globale Daten
      & string;               // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **bae_getstrpar** dient der Abfrage von im **Bartels AutoEngineer** gesetzten Stringparametern. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **bae_getdblpar**, **bae_getintpar**, **bae_setdblpar**, **bae_setintpar**, **bae_setstrpar**.

bae_inittextscreen - BAE Textbildschirm initialisieren/löschen (STD)

Synopsis

```
void bae_inittextscreen(
      );
```

Beschreibung

Die Funktion **bae_inittextscreen** löscht den Grafikarbeitsbereich für die Textausgabe. Der Textausgabecursor wird auf die linke obere Ecke gesetzt.

bae_inpoint - BAE Eingabe Punkt mit Maus (STD)

Synopsis

```
int bae_inpoint(              // Status
      double;                 // Start X-Koordinate (STD2)
      double;                 // Start Y-Koordinate (STD2)
      & double;               // Rückgabe X-Koordinate (STD2)
      & double;               // Rückgabe Y-Koordinate (STD2)
      int;                    // Zeichenmodus:
                              //    0 = keine Anzeige
                              //    1 = dynamische Rahmenanzeige
                              //    2 = dynamische Linienanzeige
                              //    3 = dynamische Kreisanzeige
                              //    4 = Gummibandabstandsanzeige
                              //    5 = Gummibandzoomfensteranzeige
                              //    6 = Gummibandquadratanzeige
                              //    7 = Gummibandquadratanzeige zentriert
                              //    8 = Gummibandfensteranzeige zentriert
                              //    9 = Gummibandpolygonlinienanzeige
                              //    10 = Gummibandumrandungspolygonanzeigey
                              //    11 = Gummibandkreisanzeige zentriert
                              //    12 = Gummiband fixiert
                              //    13 = Gummiband zentriertes Zoomfenster
                              //    14 = Keine Fensteranzeige,
                              //          RMB unmittelbarer Abbruch
                              //    15 = Anzeige mit festem Offset
      );
```

Beschreibung

Die Funktion **bae_inpoint** dient zur Wahl von Koordinaten mit der Maus. Die beiden Startkoordinaten definieren den Ausgangspunkt für Sprung relativ Kommandos. In den beiden Rückgabeparametern wird das mit Maustastendruck gewählte Koordinatenpaar zurückgegeben. Ist der Zeichenmodus 1, dann wird bei der Koordinatenwahl dynamisch ein Rahmen von den Startkoordinaten zu den aktuellen Mauskoordinaten gezeichnet. Ist der Zeichenmodus 2, dann wird während der Koordinatenwahl dynamisch eine Linie von den Startkoordinaten zu den aktuellen Mauskoordinaten gezeichnet. Der Rückgabewert ist (-1) beim Abbruch der Eingabe oder Null wenn ein Koordinatenpaar gewählt wurde.

Siehe auch

Funktion **bae_inpointmenu**.

bae_inpointmenu - BAE Eingabe Punkt mit Maus und Callbackfunktion für rechte Maustaste (STD)

Synopsis

```
int bae_inpointmenu(           // Status
      double;                  // Start X-Koordinate (STD2)
      double;                  // Start Y-Koordinate (STD2)
      & double;                // Rückgabe X-Koordinate (STD2)
      & double;                // Rückgabe Y-Koordinate (STD2)
      int;                     // Zeichenmodus:
                               //    0 = keine Anzeige
                               //    1 = dynamische Rahmenanzeige
                               //    2 = dynamische Linienanzeige
                               //    3 = dynamische Kreisanzeige
                               //    4 = Gummibandabstandsanzeige
                               //    5 = Gummibandzoomfensteranzeige
                               //    6 = Gummibandquadratanzeige
                               //    7 = Gummibandquadratanzeige zentriert
                               //    8 = Gummibandfensteranzeige zentriert
                               //    9 = Gummibandpolygonlinienanzeige
                               //    10 = Gummibandumrandungspolygonanzeigey
                               //    11 = Gummibandkreisanzeige zentriert
                               //    12 = Gummiband fixiert
                               //    13 = Gummiband zentriertes Zoomfenster
                               //    14 = nicht benutzt
                               //    15 = Anzeige mit festem Offset
      * int;                   // Callbackfunktion für rechte Maustaste
      );
```

Beschreibung

Die Funktion **bae_inpointmenu** dient zur Wahl von Koordinaten mit der Maus. Die beiden Startkoordinaten definieren den Ausgangspunkt für Sprung relativ Kommandos. In den beiden Rückgabeparametern wird das mit Maustastendruck gewählte Koordinatenpaar zurückgegeben. Ist der Zeichenmodus 1, dann wird während der Koordinatenwahl dynamisch ein Rahmen von den Startkoordinaten zu den aktuellen Mauskoordinaten gezeichnet. Ist der Zeichenmodus 2, dann wird während der Koordinatenwahl dynamisch eine Linie von den Startkoordinaten zu den aktuellen Mauskoordinaten gezeichnet. Der letzte Funktionsparameter ermöglicht die Spezifikation einer Callbackfunktion zur Aktivierung beim Drücken der rechten Maustasten (z.B. zur Anzeige spezifischer Optionsmenüs). Der Rückgabewert ist (-1) beim Abbruch der Eingabe oder Null wenn ein Koordinatenpaar gewählt wurde.

Definition der Callkbackfunktion für die rechte Maustaste

```
int callbackfunction(          // Status
      double x                 // Aktuelle Eingabe-X-Koordinate
      double y                 // Aktuelle Eingabe-Y-Koordinate
      )
{
      // Verarbeitungsprogramm
      :
      return(status);
}
```

Der Rückgabewert der Callbackfunktion sollte 0 sein zur Signalisierung der Komplettierung der Eingabe, 1 wenn die Eingabe fortzusetzen ist, oder ein beliebiger anderer Wert bei Abbruch der Eingabe. Bei Komplettierung der Eingabe (Rückgabewert 0) können geänderte Parameterwerte für die X- und Y-Koordinaten zurückgegeben werden.

Siehe auch

Funktion **bae_inpoint**.

bae_language - BAE Benutzeroberfläche Landessprache abfragen (STD)

Synopsis

```
string bae_language(          // Rückgabe Landessprachencode:
                              //    DE = deutsch
                              //    EN = englisch
                              //  für künftige Anwendungen
                              //    FR = französisch
                              //    IT = italienisch
                              //    SP = spanisch
                              //    SV = schwedisch
      );
```

Beschreibung

Die Funktion **bae_language** gibt einen Code zur Identifikation der in der BAE-Benutzeroberfläche verwendeten Landessprache zurück. Diese Information kann dazu verwendet werden, Meldungen und Menüs dynamisch an die jeweilige Landessprache anzupassen.

bae_loadcoltab - BAE Farbtabelle laden (STD)

Synopsis

```
int bae_loadcoltab(           // Status
      string;                 // Farbtabellenname
      );
```

Beschreibung

Mit der Funktion **bae_loadcoltab** wird die in der übergebenen Zeichenkette angegebene Farbtabelle geladen. Der Rückgabewert ist ungleich Null, wenn der Ladevorgang nicht erfolgreich war.

bae_loadelem - BAE Element laden (STD)

Synopsis

```
int bae_loadelem(             // Status
      string;                 // Elementdateiname
      string;                 // Elementname
      int [100,[;             // Elementklasse (STD1)
      );
```

Beschreibung

Die Funktion **bae_loadelem** lädt im **AutoEngineer**r ein Planelement in den Speicher. Im **Schaltplaneditor** sind die möglichen Werte für die Klassenangabe gegeben durch 800, 801, 802 oder 803 für SCM Stromlaufblatt, SCM Symbol, SCM Marker oder SCM Label. In den Layoutmodulen sind die zulässigen Werte für die Klassenangabe gegeben durch 100, 101, 102 oder 103 für Layout, Layout Bauteil, Layout Padstack oder Layout Pad. Im **IC-Design** System sind die zulässigen Werte für die Klassenangabe gegeben durch 1000, 1001 oder 1002 für IC-Layout, IC-Zelle oder IC-Pin. Der Rückgabewert ist Null, wenn keine Fehler aufgetreten sind, (-2) bei ungültiger Elementklassenangabe, (-1) bei Dateizugriffsfehlern, 1 wenn der Zeichensatz nicht geladen werden konnte, 2 wenn einzelne referenzierte Makros (Bibliothekselemente) nicht geladen werden konnten, oder 3 bei unplatzierten Bauteilen bzw. fehlenden Pins, d.h. wenn auf einem Layoutplan platzierte Gehäusetypen nicht mit den in der Netzliste eingetragenen Typen übereinstimmen.

Bei Angabe der Elementklasse 100 für Layouts kann wahlweise ein Leerstring für den Elementnamen übergeben werden. In diesem Fall wird automatisch das Layoutelement mit dem im System eingestellten Default-Layoutelementnamen geladen (siehe hierzu Kommando **`LAYDEFELEMENT`** für **BSETUP**).

Beim Laden von Stromlaufblättern in den **Schaltplaneditor** führt **bae_loadelem** automatisch eine **Backannotation** durch sofern dies aufgrund vorheriger Netzlistenmodifikationen im Layout erforderlich ist.

Warnung

Diese Funktion ändert sämtliche im Speicher vorhandenen Elementdaten und damit die Gültigkeit der Indexvariablen und sollte daher nur außerhalb jeglicher Programmblöcke mit Zugriff auf Indexvariablen (forall-Schleifen) aufgerufen werden. Vor Aufruf dieser Funktion sollte auf jeden Fall mit **bae_plannotsaved** nachgeprüft werden, ob das aktuelle Element mit allen Änderungen gesichert ist, da ein Undo nach **bae_loadelem** nicht möglich ist.

bae_loadfont - BAE Zeichensatz laden (STD)

Synopsis

```
int bae_loadfont(                // Status
      string;                    // Zeichensatzname
      );
```

Beschreibung

Mit der Funktion **bae_loadfont** wird der Zeichensatz mit dem in der übergebenen Zeichenkette spezifizierten Namen geladen. Der Rückgabewert ist ungleich Null, wenn der Ladevorgang nicht erfolgreich war.

bae_menuitemhelp - Onlinehilfe zu BAE-Menüelement anzeigen (STD)

Synopsis

```
int bae_menuitemhelp(            // Status
      int [0,9999];              // Menücode (STD4)
      string;                    // Help-Dateiname (Windows .hlp-Datei)
      );
```

Beschreibung

Die Funktion **bae_menuitemhelp** zeigt die Onlinehilfe zu dem über den Menücode angegebenen Topic an. Der Rückgabewert ist ungleich Null, wenn ungültige Parameter spezifiziert wurden.

Einschränkungen

bae_menuitemhelp ist nur unter Windows funktionsfähig.

bae_msgbox - BAE Info-Popup aktivieren (STD)

Synopsis

```
void bae_msgbox(
      int;                       // Info-Popup Stil/Piktogramm:
                                 //    0 = Info (Information)
                                 //    1 = Warnung (Ausrufezeichen)
                                 //    2 = Fehler (Fragezeichen)
                                 //    3 = Fataler Fehler (Stoppzeichen)
                                 //    sonst = kein Piktogramm
      string;                    // Info-Popup Text
      string;                    // Info-Popup Titel
      );
```

Beschreibung

Die Funktion **bae_msgbox** aktiviert ein Info-Popupfenster mit einer Bestätigungsabfrage (Schaltfläche OK). Auf das Erscheinungsbild des Info-Fensters kann über den ersten Parameter Einfluss genommen werden. Der angegeben Text wird innerhalb des Popupfensters angezeigt. Der Titel erscheint als Überschrift in der Titelleiste des Popupfensters. Das Erscheinungsbild bzw. das Layout des Popupfensters (Fensterposition, Titelanzeige, Schaltflächensymbole, Textausrichtung, Zeilenumbruch, usw.) kann je nach Betriebssystemplattform variieren.

Siehe auch

Funktionen **bae_msgboxverify**, **bae_msgboxverifyquit**.

bae_msgboxverify - BAE Info-Popup Ja/Nein-Abfrage aktivieren (STD)

Synopsis

```
int bae_msgboxverify(          // Rückgabe Antwortcode:
                               //    1 = Ja
                               //    0 = Nein (Default)
      string;                  // Info-Popup Text
      string;                  // Info-Popup Titel
      );
```

Beschreibung

Die Funktion **bae_msgboxverify** aktiviert ein Info-Popupfenster mit einer Ja/Nein-Abfrage. Der Funktionsrückgabewert ergibt sich zu 1 bei Eingabe bzw. Selektion von "Ja" und zu Null bei allen anderen Eingaben. Der angegeben Text wird innerhalb des Popupfensters angezeigt. Der Titel erscheint als Überschrift in der Titelleiste des Popupfensters. Das Erscheinungsbild bzw. das Layout des Popupfensters (Fensterposition, Titelanzeige, Schaltflächensymbole, Textausrichtung, Zeilenumbruch, usw.) kann je nach Betriebssystemplattform variieren.

Siehe auch

Funktionen **bae_msgbox**, **bae_msgboxverifyquit**.

bae_msgboxverifyquit - BAE Info-Popup mit Ja/Nein/Abbruch-Abfrage aktivieren (STD)

Synopsis

```
int bae_msgboxverifyquit(      // Rückgabe Antwortcode:
                               //    1 = Ja
                               //    0 = Nein
                               //    sonst = Abbruch (Default)
      string;                  // Info-Popup Text
      string;                  // Info-Popup Titel
      );
```

Beschreibung

Die Funktion **bae_msgboxverifyquit** aktiviert ein Info-Popupfenster mit einer Ja/Nein/Abbruch-Abfrage. Der Funktionsrückgabewert ergibt sich zu 1 bei Eingabe bzw. Selektion von Ja und zu Null bei Eingabe bzw. Selektion von Nein. Alle anderen Eingaben werden als Abbruchanforderung interpretiert. Der angegeben Text wird innerhalb des Popupfensters angezeigt. Der Titel erscheint als Überschrift in der Titelleiste des Popupfensters. Das Erscheinungsbild bzw. das Layout des Popupfensters (Fensterposition, Titelanzeige, Schaltflächensymbole, Textausrichtung, Zeilenumbruch, usw.) kann je nach Betriebssystemplattform variieren.

Siehe auch

Funktionen **bae_msgbox**, **bae_msgboxverify**.

bae_msgprogressrep - BAE-Fortschrittsanzeige aktivieren/aktualisieren (STD)

Synopsis

```
int bae_msgprogressrep(        // Rückgabe Status
      string;                  // Fortschrittsanzeige Text
      int [0,258];             // Fortschrittsanzeige Typ:
                               //    1 : Prozentualer Fortschritt
                               //    2 : Fortschrittsschiebebalken
                               //    |256 : Abbruchschaltfläche anzeigen
      int [0,10000];           // Fortschrittsanzeige Komplettierungsstatus [%*100]
      int [0,[;                // Fortschrittsanzeige Minimalbreite [in Zeichen]
      );
```

Beschreibung

Die Funktion **bae_msgprogressrep** aktiviert bzw. aktualisiert die BAE-Fortschrittsanzeige mit den angegebenen Parametern. Der Funktionsrückgabewert ist Null, wenn die Funktion erfolgreich ausgeführt wurde, oder ungleich Null im Fehlerfall (bei Spezifikation ungültiger Parameter). Eine mit **bae_msgprogressrep** kann mit **bae_msgprogressterm** wieder beendet werden.

Siehe auch

Funktion **bae_msgprogressterm**.

bae_msgprogressterm - BAE-Fortschrittsanzeige beenden (STD)

Synopsis

```
void bae_msgprogressterm(
      );
```

Beschreibung

Die Funktion **bae_msgprogressterm** beendet die mit **bae_msgprogressrep** aktivierte BAE-Fortschrittsanzeige.

Siehe auch

Funktion **bae_msgprogressrep**.

bae_mtpsize - BAE Popup Anzeigebereichsdimensionen abfragen (STD)

Synopsis

```
void bae_mtpsize(
      & int;                  // Rückgabe Popupmenü Textspaltenanzahl
      & int;                  // Rückgabe Popupmenü Textzeilenanzahl
      );
```

Beschreibung

Die Funktion **bae_mtpsize** ermittelt die Größe des maximal verfügbaren Anzeigebereichs zur Definition von Popupmenüs oder Toolbars mit Hilfe der Funktionen **bae_popshow** und **bae_settbsize**. Die Breite des Anzeigebereichs wird im Parameter für die Textspaltenanzahl zurückgegeben, während die Höhe des Anzeigebereichs im Parameter für die Textzeilenanzahl zurückgegeben wird. Mit Hilfe der Funktion **bae_charsize** können diese Werte in Standardlängeneinheiten umgerechnet werden.

Siehe auch

Funktionen **bae_charsize**, **bae_popshow**, **bae_settbsize**, **bae_twsize**.

bae_nameadd - BAE Namensauswahlliste Element hinzufügen (STD)

Synopsis

```
int bae_nameadd(              // Namenslistenindex/ID oder (-1) im Fehlerfall
      string;                 // Name
      string;                 // Datumsstring
      string;                 // Datumssortierstring
      int;                    // Sortiermodus:
                              //    0 = anfügen (keine Sortierung)
                              //    1 = Sortierung alphanumerisch
                              //    2 = Sortierung numerisch
                              //    3 = Sortierung nach Datum
                              //    4 = Sortierung für ID-Generierung
      );
```

Beschreibung

Die Funktion **bae_nameadd** fügt einen Eintrag in die interne BAE-Namensauswahlliste ein. Diese Liste wird von der Funktion **bae_askname** zur Aktivierung von Namensauswahldialogen benutzt. Die Funktion **bae_nameclr** kann bzw. sollte vor dem ersten Aufruf von **bae_nameadd** benutzt werden, um eventuell von vorhergehenden Applikationen vorhandene Namenseinträge aus der Namensauswahlliste zu entfernen. Die Funktion gibt den ermittelten Namenslistenindex bzw. eine ID zurück wenn die Operation erfolgreich oder (-1) im Fehlerfall.

Siehe auch
Funktionen **bae_askname**, **bae_nameclr**, **bae_nameget**.

bae_nameclr - BAE Namensauswahlliste löschen (STD)

Synopsis

```
void bae_nameclr(
      );
```

Beschreibung

Die Funktion **bae_nameclr** löscht die aktuell mit **bae_nameadd** definierte Namensliste aus dem Arbeitsspeicher.

Siehe auch

Funktionen **bae_askname**, **bae_nameadd**, **bae_nameget**.

bae_nameget - BAE Namensauswahlliste Element abfragen (STD)

Synopsis

```
int bae_nameget(                // Status
      int;                      // Namenslistenindex
      & string;                 // Rückgabe Name
      & string;                 // Rückgabe Datumsstring
      & string;                 // Rückgabe Datumssortierstring
      & string;                 // Rückgabe Kommentar
      & int;                    // Rückgabe Nameneintraganzahl oder Namens-ID
      );
```

Beschreibung

Die Funktion **bae_nameget** dient der Abfrage von zuvor mit **bae_nameadd** definierten Einträgen in der BAE-Namensauswahlliste. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder ungleich Null im Fehlerfall.

Siehe auch

Funktionen **bae_askname**, **bae_nameadd**, **bae_nameclr**.

bae_numstring - Numerische Zeichenkette erzeugen (STD)

Synopsis

```
string bae_numstring(           // Rückgabe numerische Zeichenkette
      double;                   // Eingabewert
      int [0,[;                 // Maximale Genauigkeit/Nachkommastellen
      );
```

Beschreibung

Die Funktion **bae_numstring** konvertiert den Eingabewert in eine numerische Zeichenkette mit der über die maximale Genauigkeit angegebenen Zahl von Nachkommastellen.

bae_peekiact - BAE Interaktionsvorgaben abfragen (STD)

Synopsis

```
int bae_peekiact(               // Rückgabe Interaktionsstatus/-modus:
                                //    0 = Keine automatische Interaktion anstehend
                                //    1 = Mausinteraktion anstehend
                                //    2 = Textinteraktion anstehend
                                //    3 = Menüinteraktion anstehend
                                //    4 = Tastaturinteraktion anstehen
      );
```

Beschreibung

Mit der Funktion **bae_peekiact** können die aktuellen Interaktionsvorgaben überprüft werden. Der Funktionsrückgabewert ist der Typcode der nächsten anstehenden automatischen Interaktion oder Null wenn keine automatisch Interaktion ansteht.

Siehe auch

Funktionen **bae_storekeyiact**, **bae_storemenuiact**, **bae_storemouseiact**, **bae_storetextiact**.

bae_plainmenutext - BAE Menütext konvertieren (STD)

Synopsis

```
string bae_plainmenutext(     // Zeichenkette
      string;                 // Menütext
      );
```

Beschreibung

Die Funktion **bae_plainmenutext** wandelt den über den Funktionsparameter übergebenen Menütext in normalen Text ohne Sonderzeichen für Menütrennzeilen (%), Beschleuniger (&), usw. um. Die resultierende Zeichenkette wird als Funktionsergebnis an den Aufrufer zurückgegeben.

Siehe auch

Funktionen **bae_defmenutext**, **bae_getmenutext**.

bae_planddbclass - BAE Elementklasse abfragen (STD)

Synopsis

```
int bae_planddbclass(         // Elementklasse (STD1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_planddbclass** entspricht der Datenbankklasse des aktuell geladenen Elements. Die im **Schaltplaneditor** möglichen Rückgabewerte 800, 801, 802 und 803 stehen für SCM Stromlaufblatt, SCM Symbol, SCM Marker und SCM Label. Die in den Layoutmodulen möglichen Rückgabewerte 100, 101, 102 und 103 stehen für Layout, Layout Bauteil, Layout Padstack und Layout Pad. Die im **IC-Design** möglichen Rückgabewerte 1000, 1001 und 1002 stehen für IC-Layout, IC-Zelle und IC-Pin. Der Rückgabewert (-1) gibt an, dass kein Element geladen ist.

bae_planename - BAE Elementname abfragen (STD)

Synopsis

```
string bae_planename(         // Elementname
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_planename** entspricht dem Elementnamen des aktuell geladenen Elements oder der Nullzeichenkette, wenn kein Element geladen ist.

Siehe auch

Funktion **bae_plansename**.

bae_planfname - BAE Dateiname abfragen (STD)

Synopsis

```
string bae_planfname(         // Element Dateiname
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_planfname** entspricht dem Dateinamen des aktuell geladenen Elements oder der Nullzeichenkette, wenn kein Element geladen ist.

Siehe auch
Funktionen **bae_plansfname**, **bae_setplanfname**.

bae_plannotsaved - BAE Element ungesichert Flag abfragen (STD)

Synopsis

```
int bae_plannotsaved(          // Element ungesichert Flag:
                               //    0 = Element gesichert
                               //    1 = Element nicht gesichert
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_plannotsaved** gibt an, ob das aktuell geladene Element gesichert ist. Der Wert ist ungleich Null, wenn seit dem letzten Speichern des aktuellen Elements Änderungen an diesem vorgenommen wurden.

bae_plansename - BAE Zielelementname abfragen (STD)

Synopsis

```
string bae_plansename(         // Zielelementname
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_plansename** entspricht dem Zielelementnamen des aktuell geladenen Elements oder der Nullzeichenkette, wenn kein Element geladen ist. Diese Funktion kann für Elementnamensabfragen während der Ausführung der Funktion Speichern unter benutzt werden.

Siehe auch

Funktion **bae_planename**.

bae_plansfname - BAE Zieldateiname abfragen (STD)

Synopsis

```
string bae_plansfname(         // Zieldateiname
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_plansfname** entspricht dem Zieldateinamen des aktuell geladenen Elements oder der Nullzeichenkette, wenn kein Element geladen ist. Diese Funktion kann für Elementdateinamensabfragen während der Ausführung der Funktion Speichern unter benutzt werden.

Siehe auch

Funktion **bae_planfname**.

bae_planwslx - BAE Element linke Elementgrenze abfragen (STD)

Synopsis

```
double bae_planwslx(           // Koordinatenwert (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_planwslx** entspricht der linken Begrenzungskoordinate des aktuell geladenen Elements.

bae_planwsly - BAE Element untere Elementgrenze abfragen (STD)

Synopsis

```
double bae_planwsly(           // Koordinatenwert (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_planwsly** entspricht der untereb Begrenzungskoordinate des aktuell geladenen Elements.

bae_planwsnx - BAE Element X-Bezugskoordinate abfragen (STD)

Synopsis

```
double bae_planwsnx(           // Koordinatenwert (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_planwsnx** entspricht dem X-Koordinatenbezugspunkt des aktuell geladenen Elements.

bae_planwsny - BAE Element Y-Bezugskoordinate abfragen (STD)

Synopsis

```
double bae_planwsny(           // Koordinatenwert (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_planwsny** entspricht dem Y-Koordinatenbezugspunkt des aktuell geladenen Elements.

bae_planwsux - BAE Element rechte Elementgrenze abfragen (STD)

Synopsis

```
double bae_planwsux(           // Koordinatenwert (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_planwsux** entspricht der rechten Begrenzungskoordinate des aktuell des aktuell geladenen Elements.

bae_planwsuy - BAE Element obere Elementgrenze abfragen (STD)

Synopsis

```
double bae_planwsuy(           // Koordinatenwert (STD2)
      );
```

Beschreibung
Der Rückgabewert der Funktion **bae_planwsuy** entspricht der oberen Begrenzungskoordinate des aktuell geladenen Elements.

bae_popareachoice - BAE Popupmenü Selektionsbereich definieren (STD)

Synopsis

```
int bae_popareachoice(          // Status
      double [0.0,[;            // Startzeilennummer
      double [0.0,[;            // Startspaltennummer
      double [0.0,[;            // Endzeilennummer
      double [0.0,[;            // Endspaltennummer
      string;                   // Antwort-Zeichenkette
      );
```

Beschreibung

Die Funktion **bae_popareachoice** definiert einen rechteckförmigen Selektionsbereich innerhalb des aktuell aktiven Popupmenübereichs. Der aktive Popupmenübereich wird mit der Funktion **bae_popsetarea** selektiert und ist entweder der mit **bae_popshow** definierte Popupmenübereich oder der mit **bae_settbsize** definierte Toolbarbereich. Der Selektionsbereich wird mit den Funktionsparametern für die Zeilen- bzw. Spaltenbereiche definiert, wobei Zeilen von oben nach unten und Spalten von links nach rechts gezählt werden. Mit der Funktion **bae_tbsize** kann die aktuelle Toolbargröße ermittelt werden. Mit Hilfe der Funktion **bae_charsize** können Textkoordinaten in Standardlängeneinheiten konvertiert werden. Der Parameter für die Anwort-Zeichenkette definiert den Text, der später bei Selektionen im definierten Selektionsbereich durch die Funktion **bae_readtext** zurückzugeben ist. Der Rückgabewert der Funktion **bae_popareachoice** ist (-1) im Fehlerfall, 1 bei erfolgreicher Generierung einer Schaltfläche, oder Null bei erfolgreicher Definition des Selektionsbereichs.

Siehe auch

Funktionen **bae_charsize**, **bae_popcolbar**, **bae_popcolchoice**, **bae_popsetarea**, **bae_popshow**, **bae_poptext**, **bae_poptextchoice**, **bae_readtext**, **bae_settbsize**, **bae_tbsize**.

bae_popcliparea - BAE Popupmenü Clippingbereich definieren (STD)

Synopsis

```
void bae_popcliparea(
      int [0,1];                // Clippingmodus:
                                //    0 = Clipping deaktivieren
                                //    1 = Clipping aktivieren
      double [0.0,[;            // Startzeilennummer
      double [0.0,[;            // Startspaltennummer
      double [0.0,[;            // Endzeilennummer
      double [0.0,[;            // Endspaltennummer
      );
```

Beschreibung

Die Funktion **bae_popcliparea** aktiviert (Clippingmodus 1) bzw. deaktiviert (Clippingmodus 0) das sogenannte Clipping innerhalb des aktuell aktiven Popupmenübereichs. Der aktive Popupmenübereich wird mit der Funktion **bae_popsetarea** selektiert und ist entweder der mit **bae_popshow** definierte Popupmenübereich oder der mit **bae_settbsize** definierte Toolbarbereich. Mit dem Clippingverfahren können Grafikausgaben in den Popupmenüs bzw. Toolbars automatisch auf spezielle Teilbereiche der Anzeige beschränkt werden, was insbesondere bei der Generierung von Ausschnittszeichnungen mit Hilfe der Funktion **bae_popdrawpoly** von besonderem Nutzen ist. Der Clippingbereich wird mit den Funktionsparametern für die Zeilen- bzw. Spaltenbereiche definiert, wobei Zeilen von oben nach unten und Spalten von links nach rechts gezählt werden. Mit Hilfe der Funktion **bae_charsize** können Textkoordinaten in Standardlängeneinheiten konvertiert werden.

Siehe auch
Funktionen **bae_charsize**, **bae_popdrawpoly**, **bae_popsetarea**, **bae_popshow**, **bae_settbsize**.

bae_popclrtool - BAE Toolbar-Popupmenübereich löschen (STD)

Synopsis

```
void bae_popclrtool(
      );
```

Beschreibung

Mit der Funktion **bae_popclrtool** können sämtliche Anzeigen im aktuellen Toolbarbereich gelöscht bzw. deaktiviert werden.

Siehe auch

Funktionen **bae_popsetarea**, **bae_settbsize**.

bae_popcolbar - BAE Popupmenü Farbbalkenanzeige definieren (STD)

Synopsis

```
int bae_popcolbar(            // Status
      double [0.0,[;          // Startzeilennummer
      double [0.0,[;          // Startspaltennummer
      double [0.0,[;          // Endzeilennummer
      double [0.0,[;          // Endspaltennummer
      int [0,[;               // Farbwert (STD18)
      );
```

Beschreibung

Die Funktion **bae_popcolbar** definiert einen nicht selektierbaren Farbbalken innerhalb des zuvor mit **bae_popshow** aktivierten Popupmenüs. Die Größe und die Position für die Anzeige des Farbbalkens ergeben sich aus den übergebenen Zeilen- und Spaltenparametern; die Koordinate [0,0] bezieht sich dabei auf die linke obere Ecke des Popupbereiches. Der Farbwert-Parameter definiert die Farbe des Farbbalkens. Der Rückgabewert der Funktion ist ungleich Null, wenn fehlerhafte Parameter übergeben wurden.

Siehe auch

Funktionen **bae_popareachoice**, **bae_popcolchoice**, **bae_popshow**, **bae_poptext**, **bae_poptextchoice**.

bae_popcolchoice - BAE Popupmenü Farbbalkenselektion definieren (STD)

Synopsis

```
int bae_popcolchoice(         // Status
      double [0.0,[;          // Startzeilennummer
      double [0.0,[;          // Startspaltennummer
      double [0.0,[;          // Endzeilennummer
      double [0.0,[;          // Endspaltennummer
      int [0,[;               // Farbwert (STD18)
      string;                 // Antwort-Zeichenkette
      );
```

Beschreibung

Die Funktion **bae_popcolchoice** definiert einen maus-selektierbaren Farbbalken innerhalb des zuvor mit **bae_popshow** aktivierten Popupmenüs. Die Größe und die Position für die Anzeige des Farbbalkens ergeben sich aus den übergebenen Zeilen- und Spaltenparametern; die Koordinate [0,0] bezieht sich dabei auf die linke obere Ecke des Popupbereiches. Der Farbwert-Parameter definiert die Farbe des Farbbalkens. Der Rückgabewert der Funktion ist ungleich Null, wenn fehlerhafte Parameter übergeben wurden. Die Selektion des mit **bae_popcolchoice** definierten Farbbalkens ist nach einer Aktivierung der Funktion **bae_readtext** möglich. Durch die Selektion des Farbbalkens wird die Funktion **bae_readtext** beendet; der Rückgabewert von **bae_readtext** ergibt sich dabei automatisch aus der Antwort-Zeichenkette, die beim Aufruf der Funktion **bae_popcolchoice** angegeben wurde.

Siehe auch

Funktionen **bae_popareachoice**, **bae_popcolbar**, **bae_popshow**, **bae_poptext**, **bae_poptextchoice**, **bae_readtext**.

bae_popdrawpoly - BAE Popupmenü Polygon-/Grafikanzeige (STD)

Synopsis

```
int bae_popdrawpoly(           // Status
      int [0,[;                // Polygon Farbe (STD18)
      int [0,3];               // Polygon Zeichenmodus (STD19)
      int [0,15];              // Polygon Füllmodus (STD20)
      );
```

Beschreibung

Die Funktion **bae_popdrawpoly** bildet das mit **bae_storepoint** generierte interne Polygon in der angegebenen Farbe und unter Berücksichtigung der spezifizierten Parameter für den Zeichen- und Füllmodus im aktuell aktiven Popupmenübereich ab. Der aktive Popupmenübereich wird mit der Funktion **bae_popsetarea** selektiert und ist entweder der mit **bae_popshow** definierte Popupmenübereich oder der mit **bae_settbsize** definierte Toolbarbereich. Die Polygonanzeige kann mit Hilfe der Funktion **bae_popcliparea** auf einen speziellen Teilbereich des Popupmenüs beschränkt werden. Mit Hilfe der Funktion **bae_popareachoice** können ein oder mehrere Teilbereiche der Polygonanzeige für eine spätere Selektion durch die Funktion **bae_readtext** vorgesehen werden. Zur Vorbereitung der Definition neuer Polygone kann das aktuell generierte, interne Polygon mit der Funktion **bae_clearpoints** gelöscht werden. Die Basislänge für die Ausgabe gestrichelter Linienzüge kann mit Hilfe der Funktion **bae_setpopdash** gesetzt werden. Der Rückgabewert der Funktion **bae_popdrawpoly** ist Null, wenn die Polygongenerierung erfolgreich durchgeführt wurde, oder ungleich Null im Fehlerfall.

Siehe auch

Funktionen **bae_clearpoints**, **bae_dialbmpalloc**, **bae_popareachoice**, **bae_popcliparea**, **bae_popcolbar**, **bae_popcolchoice**, **bae_popdrawtext**, **bae_popsetarea**, **bae_popshow**, **bae_readtext**, **bae_setpopdash**, **bae_settbsize**, **bae_storepoint**.

bae_popdrawtext - BAE Popupmenü Textanzeige (STD)

Synopsis

```
int bae_popdrawtext(           // Status
      int;                     // Text Zeile
      int;                     // Text Spalte
      int [0,[;                // Text Farbe (STD18)
      int [0,[;                // Text Hintergrundfarbe (STD18)
      string;                  // Text String
      );
```

Beschreibung

Die Funktion **bae_popdrawtext** zeigt den angegebenen Text an den spezifizierten Zeilen- und Spaltenkoordinaten im aktuell aktiven Popupmenübereich an, wobei zur Darstellung die angegebenen Farben für den Text bzw. den Texthintergrund verwendet werden. Der aktive Popupmenübereich wird mit der Funktion **bae_popsetarea** selektiert und ist entweder der mit **bae_popshow** definierte Popupmenübereich oder der mit **bae_settbsize** definierte Toolbarbereich. Mit Hilfe der Funktion **bae_charsize** können Textkoordinaten in Standardlängeneinheiten konvertiert werden. Mit Hilfe der Funktion **bae_popareachoice** können ein oder mehrere Teilbereiche der Textanzeige für eine spätere Selektion durch die Funktion **bae_readtext** vorgesehen werden. Der Rückgabewert der Funktion **bae_popdrawtext** ist Null, wenn die Textgenerierung erfolgreich durchgeführt wurde, oder ungleich Null im Fehlerfall.

Siehe auch

Funktionen **bae_charsize**, **bae_dialbmpalloc**, **bae_popareachoice**, **bae_popdrawpoly**, **bae_popsetarea**, **bae_popshow**, **bae_readtext**, **bae_settbsize**.

bae_popmouse - BAE Popup/Toolbar Mausposition abfragen (STD)

Synopsis

```
void bae_popmouse(
      & double;              // Rückgabe Maus-X-Koordinate/Spalte
      & double;              // Rückgabe Maus-Y-Koordinate/Reihe
      & int;                 // Rückgabe Mausstatus:
                             //    Bit 1 : linke Maustaste gedrückt
                             //    Bit 2 : rechte Maustaste gedrückt
                             //    Bit 3 : mittlere Maustaste gedrückt
      );
```

Beschreibung

Mit der Funktion **bae_popmouse** können die aktuellen Mauskoordinaten (Spalte und Reihe) innerhalb des Popup- bzw. Toolbarbereichs ermittelt werden. Der Parameter zur Rückgabe des Mausstatus gibt an, welche Maustasten gerade gedrückt sind.

Siehe auch

Funktion **bae_wsmouse**.

bae_poprestore - BAE Popupmenübereich reaktivieren (STD)

Synopsis

```
void bae_poprestore(
      );
```

Beschreibung

Die Funktion **bae_poprestore** deaktiviert das zuvor mit **bae_popshow** aktivierte Popupmenü und reaktiviert ggf. wieder die Anzeige des durch das Popupmenü überdeckten Grafikarbeitsbereiches.

Siehe auch

Funktion **bae_popshow**.

bae_popsetarea - BAE Toolbar-Popupmenübereich löschen (STD)

Synopsis

```
void bae_popsetarea(
      int                    // Popupbereich:
                             //    0 = Popupmenübereich
                             //    1 = Toolbarbereich
                             //    2..31 = Dialogbitmapbereich
      );
```

Beschreibung

Die Funktion **bae_popsetarea** dient dazu, wahlweise das Standard-Popupmenü (Popupbereich 0), die Toolbar (Popupbereich 1) oder eine Dialogbitmap (Popupbereich bzw. Popupbitmapnummer 2 bis 31) für nachfolgende Popupoperationen wie z.B. **bae_popareachoice**, **bae_popcliparea**, **bae_popdrawpoly** oder **bae_popdrawtext** zu aktivieren. Die Definition bzw. Anzeige des Standard-Popupmenübereichs erfolgt mit Hilfe der Funktion **bae_popshow**. Die Definition bzw. Anzeige des Toolbar-Popupmenübereichs erfolgt mit Hilfe der Funktion **bae_settbsize**. Wenn mit **bae_popsetarea** kein Popupmenübereich explizit aktiviert wurde, dann ist zunächst das Standard-Popupmenü für Popupoperationen selektiert.

Siehe auch

Funktionen **bae_dialbmpalloc**, **bae_popareachoice**, **bae_popcliparea**, **bae_popdrawpoly**, **bae_popdrawtext**, **bae_popshow**, **bae_settbsize**.

bae_popshow - BAE Popupmenü aktivieren (STD)

Synopsis

```
int bae_popshow(                 // Status
      & double [0.0,[;          // Popup-Zeilenanzahl
      & double [0.0,[;          // Popup-Spaltenanzahl
      & double [0,1.0];         // Linke Popup-Begrenzung
      & double [0,1.0];         // Untere Popup-Begrenzung
      & double [0,1.0];         // Rechte Popup-Begrenzung
      & double [0,1.0];         // Obere Popup-Begrenzung
      );
```

Beschreibung

Die Funktion **bae_popshow** erzeugt ein Popupmenü in der angegebenen Größe. Die Position sowie die maximale Ausdehnung des Popupmenüs ergeben sich aus den Popup-Begrenzungsparametern; die angegebenen Werte verstehen sich dabei als Relativwerte in Bezug auf die Größe des zur Verfügung stehenden Grafikarbeitsbereiches; der Wert 0.0 steht dabei für Minimalgröße, der Wert 1.0 für Maximalgröße. Die Anzahl der maximal anzeigbaren Spalten und Zeilen kann mit der Funktion **bae_mtpsize** ermittelt werden. Die Parameter für die Popupzeilen- und Popupspaltenanzahl definieren als Eingabeparameter die gewünschte Größe des Popupmenüs. Die innerhalb der angegebenen Popupbegrenzung tatsächlich darstellbaren Zeilen und Spalten werden durch die Funktion **bae_popshow** automatisch berechnet und in den entsprechenden Parametern an den Aufrufer wieder zurückgegeben. Der Rückgabewert der Funktion **bae_popshow** ist ungleich Null, wenn ungültige oder sich widersprechende Parameter übergeben wurden. Die Menüfarben für Texte, Hintergrund und Rahmen des Popups ergeben sich aus den mit dem Utilityprogramm **BSETUP** im BAE Setup entsprechend eingetragenen Farbwerten. Nach dem Aufruf der Funktion **bae_popshow** können mit den Funktionen **bae_popareachoice**, **bae_popcolbar**, **bae_popcolchoice**, **bae_popdrawpoly**, **bae_popdrawtext**, **bae_poptext** und **bae_poptextchoice** selektierbare und nicht selektierbare Farbbalken, Anzeigetexte, Grafiken und Selektionsbereiche definiert werden. Die Popupmenüauswahl kann anschließend mit der Funktion **bae_readtext** aktiviert werden.

Warnung

Zur Freigabe eines durch ein Popupmenü belegten Grafikarbeitsbereiches ist die Funktion **bae_poprestore** zu benutzen.

Siehe auch

Funktionen **bae_mtpsize**, **bae_popareachoice**, **bae_popcliparea**, **bae_popcolbar**, **bae_popcolchoice**, **bae_popdrawpoly**, **bae_popdrawtext**, **bae_poprestore**, **bae_popsetarea**, **bae_poptext**, **bae_poptextchoice**, **bae_readtext**, **bae_settbsize** sowie BAE Utilityprogramm **BSETUP**.

bae_poptext - BAE Popupmenü Textanzeige definieren (STD)

Synopsis

```
int bae_poptext(                 // Status
      double [0.0,[;            // Zeilennummer
      double [0.0,[;            // Spaltennummer
      string;                   // Anzeige-Zeichenkette
      );
```

Beschreibung

Die Funktion **bae_poptext** definiert einen nicht selektierbaren Text innerhalb des zuvor mit **bae_popshow** aktivierten Popupmenüs. Die Position für die Anzeige des Textes ergibt sich aus den übergebenen Zeilen- und Spaltenparametern; die Koordinate [0,0] bezieht sich dabei auf die linke obere Ecke des Popupbereiches. Der im Popupmenü anzuzeigende Text wird durch die Anzeige-Zeichenkette definiert. Der Rückgabewert der Funktion ist ungleich Null, wenn fehlerhafte Parameter übergeben wurden.

Siehe auch

Funktionen **bae_popcolbar**, **bae_popcolchoice**, **bae_popshow**, **bae_poptextchoice**.

bae_poptextchoice - BAE Popupmenü Textselektion definieren (STD)

Synopsis

```
int bae_poptextchoice(          // Status
      double [0.0,[;            // Zeilennummer
      double [0.0,[;            // Spaltennummer
      string;                   // Anzeige-Zeichenkette
      string;                   // Antwort-Zeichenkette
      );
```

Beschreibung

Die Funktion **bae_poptextchoice** definiert einen maus-selektierbaren Text innerhalb des zuvor mit **bae_popshow** aktivierten Popupmenüs. Die Position für die Anzeige des Textes ergibt sich aus den übergebenen Zeilen- und Spaltenparametern; die Koordinate [0,0] bezieht sich dabei auf die linke obere Ecke des Popupbereiches. Der im Popupmenü anzuzeigende Text wird durch die Anzeige-Zeichenkette definiert. Der Rückgabewert der Funktion ist ungleich Null, wenn fehlerhafte Parameter übergeben wurden. Die Selektion der mit **bae_poptextchoice** definierten Textanzeige ist nach einer Aktivierung der Funktion **bae_readtext** möglich. Durch die Selektion des Anzeigetextes wird die Funktion **bae_readtext** beendet; der Rückgabewert von **bae_readtext** ergibt sich dabei automatisch aus der Antwort-Zeichenkette, die beim Aufruf der Funktion **bae_poptextchoice** angegeben wurde.

Siehe auch

Funktionen **bae_popcolbar**, **bae_popcolchoice**, **bae_popshow**, **bae_poptext**, **bae_readtext**.

bae_postprocess - BAE Postprozessorlauf (STD)

Synopsis

```
void bae_postprocess(
      );
```

Beschreibung

Die Funktion **bae_postprocess** führt einen BAE-Prostprozesslauf für das aktuell geladene Element durch. Auf Layoutebene wird damit die eine Aktualisierung der Connectivity und eine Prüfung der Designregeln erzwungen.

bae_progdir - BAE Programmverzeichnis ermitteln (STD)

Synopsis

```
string bae_progdir(             // BAE-Programmverzeichnis-Name
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_progdir** entspricht dem aktuell gültigen Namen für das BAE-Programmverzeichnis. Diese Information ist insbesondere dann von Nutzen, wenn auf spezielle Daten aus dem BAE-Programmverzeichnis zugegriffen werden soll (z.B. zum Laden von Farbtabellen, für den Zugriff auf Blendentabellen, usw.).

bae_prtdialog - BAE Dialogtextausgabe in Statuszeile (STD)

Synopsis

```
void bae_prtdialog(
      string;                   // Anzeige-Zeichenkette
      );
```

Beschreibung

Die Funktion **bae_prtdialog** gibt die übergebene Anzeige-Zeichenkette in der Statuszeile der BAE-Benutzeroberfläche aus.

Siehe auch

Funktion **perror**.

bae_querydist - BAE Punkt-zu-Polgygon Distanzabfrage (STD)

Synopsis

```
int bae_querydist(             // Status
      double;                  // Abfrage X-Koordinate (STD2)
      double;                  // Abfrage Y=Koordinate (STD2)
      & double;                // Rückgabe Distanz (STD2)
      );
```

Beschreibung

Mit der Funktion **bae_querydist** kann der Abstand zwischen dem über die Parameter für die X- und Y-Koordinaten gegebenen Punkt und das zuvor mit **bae_storedistpoly** gespeicherte interne Distanzabfragepolygon ermittelt werden. Die ermittelte Distanz wird im letzten Funktionsparameter zurückgegeben. Positive Werte geben dabei Abstände zu Punkten außerhalb des Distanzabfragepolygons an, negative Werte werden für Abstände zu Punkten innerhalb des Polygons zurückgegeben. Der Funktionsrückgabewert ist Null bei erfolgreicher Distanzabfrage oder ungleich Null wenn ein Fehler aufgetreten ist (fehlendes Distanzabfragepolygon).

Siehe auch

Funktion **bae_storedistpoly**.

bae_readedittext - BAE Texteingabe/-anzeige (STD)

Synopsis

```
string bae_readedittext(       // Zeichenkette
      string;                  // Eingabeaufforderung
                               //    !-Präfix: Mehrzeilentexteingabe
                               //    sonst: Einzelzeilentexteingabe
      string;                  // Default-Rückgabezeichenkette
      int [0,[;                // Maximale Eingabelänge
      );
```

Beschreibung

Die Funktion **bae_readedittext** aktiviert einen Dialog zum Anzeigen und Editieren von Texten. Die über den ersten Funktionsparameter angegebene Eingabeaufforderung wird in der Titelleiste des Dialogs angezeigt. Ist das erste Zeichen der Eingabeaufforderung ein Ausrufungszeichen (**!**), dann wird ein Dialog mit einem mehrzeiligem Editierfenster aktiviert, ansonsten erfolgt die Eingabe über einen Dialog mit einem einzeiligen Editierfenster. Das Dialogfenster für mehrzeilige Texteingaben ist in der Größe veränderbar. Nach dem Aufruf der Funktion wird im Editierfenster die angegebene Rückgabezeichenkette angezeigt. Der Dialog enthält Schaltflächen zur Bestätigung (OK) und zum Abbruch (Abbruch) der Texteingabe. Im Dialog für die mehrzeilige Texteingabe werden außerdem Schaltflächen zum Laden des Inhalts einer selektierbaren Datei (Laden) und zum Speichern des aktuell editierten Texts in eine Ausgabedatei (Speichern) angeboten. Der Rückgabewert dieser Funktion entspricht der vom Benutzer editierten Zeichenkette wenn die Eingabe mit OK bestätigt wurde. Dabei ist die eingebbare maximale Länge der Antwortzeichenkette durch den entsprechenden Funktionsparameter festgelegt. Bei Betätigung von Abbruch wird die Default-Rückgabezeichenkette an den Aufrufer zurückgegeben.

Warnung

Unter **BAE Demo** steht die Option zum Speichern des aktuell bearbeiteten Texts nicht zur Verfügung, und die üblicherweise über die rechte Maustaste unter Windows verfügbare Funktion zum Kopieren des aktuell bearbeiteten Texts in das Clipboard ist ebenfalls deaktiviert.

Siehe auch

Funktion **bae_readtext**.

bae_readtext - BAE Texteingabe mit optionalem Popupmenü (STD)

Synopsis

```
string bae_readtext(           // Zeichenkette
      string;                  // Eingabeaufforderung
      int;                     // Maximale Eingabelänge
      );
```

Beschreibung

Die Funktion **bae_readtext** fordert den Benutzer in der Eingabezeile mit der übergebenen Promptzeichenkette zur Eingabe einer Zeichenkette auf. Der Rückgabewert dieser Funktion entspricht der vom Benutzer eingegebenen Zeichenkette, wobei die über Tastatur eingebbare maximale Länge der Antwortzeichenkette durch den entsprechenden Funktionsparameter definiert wird. Wurde vor dem **bae_readtext**-Aufruf mit **bae_setmousetext** die Möglichkeit der Textübergabe per Mausklick aktiviert, dann ist parallel zur Tastatureingabe die Betätigung einer Maustaste möglich; der Rückgabewert der Funktion **bae_readtext** ergibt sich dann aus der mit **bae_setmousetext** definierten Antwortzeichenkette. Nach Ablauf der Funktion **bae_readtext** wird die mit **bae_setmousetext** definierte Möglichkeit der Mausklickeingabe wieder deaktiviert. Wurde vor dem **bae_readtext**-Aufruf mit **bae_popshow** ein Popupmenü aktiviert (ohne dass eine Mausklickeingabe mit **bae_setmousetext** aktiviert wurde), dann ist parallel zur Tastatureingabe die Mausselektion eines der innerhalb des Popupmenüs mit **bae_popcolchoice** bzw. mit **bae_poptextchoice** definierten Selektionselemente (Farbbalken bzw. Texte) möglich, wobei sich dann die entsprechend definierten Zeichenketten-Rückgabewerte ergeben.

Warnung

Die Funktion **bae_readtext** deaktiviert alle zuvor mit **bae_popcolchoice**, **bae_poptextchoice** und **bae_setmousetext** aktivierten Selektionsalternativen.

Siehe auch

Funktionen **bae_popcolchoice**, **bae_popshow**, **bae_poptextchoice**, **bae_readedittext**, **bae_setmousetext**.

bae_redefmainmenu - BAE Hauptmenüdefinition starten (STD)

Synopsis

```
int bae_redefmainmenu(         // Status
      );
```

Beschreibung

Die Funktion **bae_redefmainmenu** startet die (Neu-)Definition des Hauptmenüs im aktuell aktiven BAE-Modul. Der Rückgabewert der Funktion **bae_redefmainmenu** ist (-1), wenn ein Fehler aufgetreten ist oder Null andernfalls. Im Anschluss an den Aufruf der Funktion **bae_redefmainmenu** sollten über die Funktion **bae_defmenu** (zu beenden mit **bae_endmenu**) zumindest die Hauptmenüeinträge definiert werden. Wahlweise können anschließend auch die Untermenüs über die Funktion **bae_defselmenu** (zu beenden mit **bae_endmenu**) konfiguriert werden. Zwischen den Funktionsaufrufen für **bae_defmenu** bzw. **bae_defselmenu** einerseits und der Funktion **bae_endmenu** sind mit durch wiederholten Aufruf der Funktion **bae_defmenutext** die einzelnen Menüeinträge zu definieren. Die Definition des Hauptmenüs *muss* mit der Funktion **bae_endmainmenu** beendet werden. Mit der Funktion **bae_resetmenuprog** können *sämtliche* Menübelegungen wieder zurückgesetzt werden.

Siehe auch

Funktionen **bae_defmenu**, **bae_defmenuprog**, **bae_defmenutext**, **bae_defselmenu**, **bae_endmainmenu**, **bae_endmenu**, **bae_redefmenu**, **bae_resetmenuprog**.

bae_redefmenu - BAE Menüfunktion umprogrammieren (STD)

Synopsis

```
int bae_redefmenu(            // Status
      int [0,999];            // Menünummer
      int [0,99];             // Menüzeile
      string;                 // Menütext
      int;                    // BAE-Menüfunktion (STD4)
      int;                    // Menüeintrag Bearbeitungsschlüssel:
                              //    8000000h = immer aktivierbar
                              //    7FFFFFFh = für jeden Elementtyp
                              //                aktivierbar
                              //    sonstige = (kombinierter)
                              //                DDB-Klassen-Schlüssel
      );
```

Beschreibung

Die Funktion **bae_redefmenu** ordnet einem Menüeintrag den angegebenen Menütext sowie die numerisch spezifizierte BAE-Menüfunktion zu. Die Menünummer gibt die Nummer des Hauptmenüs an, die Menüzeile die Position im zugehörigen Submenü. Der Bearbeitungsschlüssel dient der Konfiguration von sogenannten Ghostmenüs. Hierfür ist ein kodierter Integerwert einzutragen, wie er mit den Funktionen **bae_getclassbitfield** oder **bae_getmenubitfield** ermittelt bzw. definiert werden kann (im Zweifelsfall empfiehlt sich die Verwendung des Hexadezimalwertes 80000000h, um sicherzustellen, dass die Funktion immer aktivierbar ist). Der Rückgabewert der Funktion **bae_redefmenu** ist (-1), wenn ein Fehler aufgetreten ist oder Null andernfalls. Mit der Funktion **bae_resetmenuprog** können *sämtliche* Menübelegungen wieder zurückgesetzt werden.

Siehe auch

Funktionen **bae_defmenuprog**, **bae_getclassbitfield**, **bae_getmenubitfield**, **bae_resetmenuprog**.

bae_resetmenuprog - BAE Menüprogrammierung zurücksetzen (STD)

Synopsis

```
void bae_resetmenuprog(
      );
```

Beschreibung

Die Funktion **bae_resetmenuprog** setzt *sämtliche* mit **bae_deffuncprog**, **bae_defkeyprog**, **bae_defmenuprog**, **bae_redefmainmenu**, **bae_defmenu**, **bae_defselmenu**, **bae_defmenutext**, **bae_redefmenu** vorgenommenen Tasten- und Menüdefinitionen zurück und versetzt das System so in den Default Anfangszustand.

Siehe auch

Funktionen **bae_deffuncprog**, **bae_defkeyprog**, **bae_defmenu**, **bae_defmenuprog**, **bae_defmenutext**, **bae_defselmenu**, **bae_redefmainmenu**, **bae_redefmenu**.

bae_sendmsg - BAE HighEnd Message senden (STD/HighEnd)

Synopsis

```
int bae_sendmsg(                // Status
      string;                   // Meldung Textstring
      int [0,1];                // Flag - projektspezifische Nachricht
      );
```

Beschreibung

Die Funktion **bae_sendmsg** ist nur in **BAE HighEnd** verfügbar. Wenn die Funktion nicht in **BAE HighEnd** aufgerufen wird, oder wenn Parameterangaben ungültig oder unvollständig sind, dann wird ein Wert ungleich Null zurückgegeben. Mit **bae_sendmsg** wird ein Messagestring an die anderen Module eine **BAE HighEnd**-Sitzung übermittelt. Dabei kann über den zweiten Funktionsparameter angegeben werden, ob alle Module die Nachricht erhalten sollen, oder ob die Nachricht nur an die Module geschickt werden soll, in denen gerade Elemente derselben DDB-Datei bearbeitet werden. Zu einer Sitzung gehören alle Programminstanzen, die ausgehend von einem BAE-Aufruf über die Funktion Weiteres Fenster aus dem BAE-Hauptmenü oder dem **Schaltplaneditor** gestartet wurden. In den addressierten Modulen wird bei Ankunft einer Nachricht automatisch das **User Language**-Programm **bae_msg** gestartet. Ist **bae_msg** nicht verfügbar, dann erfolgt der Aufruf eines **User Language**-Programms mit modulspezifischem Namen (**scm_msg** im **Schaltplaneditor**, **ged_msg** im **Layouteditor**, **ar_msg** im **Autorouter**, etc.). Das automatisch gestartete **User Language**-Programm muss die Nachricht mit der Funktion **bae_getmsg** entgegennehmen. Die Nachricht steht nur während der Dauer dieses Programmaufrufs zur Verfügung. Wird die Nachricht von dem ***_msg**-Programm nicht angenommen, dann geht sie verloren. Der Inhalt der Nachricht kann dazu benutzt werden, eine spezielle Aktion im empfangenden Modul auszulösen.

Siehe auch

Funktion **bae_getmsg**.

bae_setanglelock - BAE Winkelfreigabeflag setzen (STD)

Synopsis

```
int bae_setanglelock(           // Status
      int [0,1];                // Winkelfreigabe Flag (STD9)
      );
```

Beschreibung

Die Funktion **bae_setanglelock** setzt den Wert des Winkelfreigabeflags im **AutoEngineer** (0=Winkel freigeben, 1=Winkel einhalten). Es wird ein Wert ungleich Null zurückgegeben, wenn ein ungültiger Flagwert angegeben wurde.

Siehe auch

Funktion **bae_getanglelock**.

bae_setbackgrid - BAE Hintergrundraster setzen (STD)

Synopsis

```
int bae_setbackgrid(            // Status
      double [0.0,[;            // X-Hintergrundraster (STD2)
      double [0.0,[;            // Y-Hintergrundraster (STD2)
      );
```

Beschreibung

Die Funktion **bae_setbackgrid** setzt die Werte für das Hintergrundraster im **AutoEngineer**. Werte von Null spezifizieren, dass kein Hintergrundraster angezeigt werden soll. Es wird ein Wert ungleich Null zurückgegeben, wenn ungültige Rasterwerte spezifiziert wurden.

Siehe auch

Funktion **bae_getbackgrid**.

bae_setclipboard - Textstring in (Windows-)Zwischenablage speichern (STD)

Synopsis

```
int bae_setclipboard(          // Status
      string;                  // Textstring
      );
```

Beschreibung

Die Funktion **bae_setclipboard** speichert den ügergebenen Textstring in der **Windows**-Zwischenablage. Der Funktionsrückgabewert ist Null bei erfolgreicher Ausführung der Funktion oder ungleich Null im Fehlerfall.

bae_setcolor - BAE Farbwert setzen (STD)

Synopsis

```
int bae_setcolor(              // Status
      int;                     // Anzeigeelementtyp (SCM1|LAY9|ICD9)
      int;                     // Farbwert (STD18)
      );
```

Beschreibung

Die Funktion **bae_setcolor** setzt den Farbwert für den angegebenen Anzeigeelementtyp. Der Wert für den Anzeigeelementtyp muss entsprechend der aktuellen Interpreterumgebung gesetzt sein. Der Rückgabewert ist ungleich Null, wenn der Farbwert nicht gesetzt werden konnte.

Warnung

Um beim Umdefinieren von Farbtabellen unnötige Bildneuaufbauten zu vermeiden, führt die Funktion **bae_setcolor** keinen automatischen Bildneuaufbau durch. Es liegt somit in der Verantwortung des Aufrufers, den erforderlichen Bildneuaufbau nach einer mit **bae_setcolor** durchgeführten Farbtabellendefinitionssequenz auszulösen.

Siehe auch

Funktion **bae_getcolor**.

bae_setcoorddisp - BAE Koordinatenanzeige setzen (STD)

Synopsis

```
int bae_setcoorddisp(          // Status
      int [0,1];               // Koordinatenanzeigemodus (STD7)
      );
```

Beschreibung

Die Funktion **bae_setcoorddisp** setzt den Koordinatenanzeigemodus im **AutoEngineer**, wobei der Wert 0 für mm-Einheiten (bzw. Mikrometer-Einheiten im **IC-Design** System) und der Wert 1 für Inch-Einheiten (bzw. mil-Einheiten im **IC-Design** System) anzugeben ist. Es wird ein Wert ungleich Null zurückgegeben, wenn ein ungültiger Modus angegeben wurde.

Siehe auch
Funktion **bae_getcoorddisp**.

bae_setdblpar - BAE Doubleparameter setzen (STD)

Synopsis

```
int bae_setdblpar(              // Status
      int [0,[;                 // Parametertyp/-nummer:
                                //    0 = maximale Dialogboxbreite
                                //    1 = maximale Dialogboxhöhe
                                //    2 = Grafikanzeige Zoomfaktor
                                //    3 = Gummibandeckradius (STD2)
                                //    4 = Gummiband X-Vektorkoordinate (STD2)
                                //    5 = Gummiband Y-Vektorkoordinate (STD2)
                                //    6 = Fixierte X-Pickkoordinate (STD2)
                                //    7 = Fixierte Y-Pickkoordinate (STD2)
                                //  [ 8 = Systemparameter - kein Schreibzugriff ]
                                //  [ 9 = Systemparameter - kein Schreibzugriff ]
                                //    10 = Bildschirmpickbereich (STD2)
                                //    11 = Elementauswahl relativer
                                //          Vorschaubereich [0.05, 0.95]
                                //  [ 12 = Systemparameter - kein Schreibzugriff ]
                                //  [ 13 = Systemparameter - kein Schreibzugriff ]
      double;                   // Parameterwert
      );
```

Beschreibung

Die Funktion **bae_setdblpar** dient dazu, Systemparameter vom Typ **double** im **Bartels AutoEngineer** zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **bae_setdblpar** gesetzten Systemparametern können mit der Funktion **bae_getdblpar** abgefragt werden.

Siehe auch

Funktionen **bae_getdblpar**, **bae_getintpar**, **bae_getstrpar**, **bae_setintpar**, **bae_setstrpar**.

bae_setgridlock - BAE Rasterfreigabeflag setzen (STD)

Synopsis

```
int bae_setgridlock(            // Status
      int [0,1];                // Rasterfreigabe Flag (STD8)
      );
```

Beschreibung

Die Funktion **bae_setgridlock** setzt den Wert des Rasterfreigabeflags im **AutoEngineer** (0=Raster freigeben, 1=Raster einhalten). Es wird ein Wert ungleich Null zurückgegeben, wenn ein ungültiger Flagwert angegeben wurde.

Siehe auch

Funktion **bae_getgridlock**.

bae_setgridmode - BAE Rasterabhängigkeitsmodus setzen (STD)

Synopsis

```
int bae_setgridmode(           // Status
      int [0,255];             // Modus für automatische Rastereinstellung:
                               //    0x01: Eingaberaster = 0.25 × Hintergrundraster
                               //    0x02: Eingaberaster = 0.50 × Hintergrundraster
                               //    0x04: Eingaberaster = 1.00 × Hintergrundraster
                               //    0x08: Eingaberaster = 2.00 × Hintergrundraster
                               //    0x10: Hintergrundraster = 0.25 × Eingaberaster
                               //    0x20: Hintergrundraster = 0.50 × Eingaberaster
                               //    0x40: Hintergrundraster = 1.00 × Eingaberaster
                               //    0x80: Hintergrundraster = 2.00 × Eingaberaster
      );
```

Beschreibung

Die Funktion **bae_setgridmode** setzt den BAE Rasterabhängigkeitsmodus. Der Funktionsrückgabewert ist ungleich Null wenn ein ungültiger Rasterabhängigkeitsmodus spezifiziert wurde.

Siehe auch

Funktion **bae_getgridmode**.

bae_setinpgrid - BAE Eingaberaster setzen (STD)

Synopsis

```
int bae_setinpgrid(            // Status
      double [0.0,[;           // X-Eingaberaster (STD2)
      double [0.0,[;           // Y-Eingaberaster (STD2)
      );
```

Beschreibung

Die Funktion **bae_setinpgrid** setzt die Werte für das Eingaberaster im **AutoEngineer**. Es wird ein Wert ungleich Null zurückgegeben, wenn ungültige Rasterwerte spezifiziert wurden.

Siehe auch

Funktion **bae_getinpgrid**.

bae_setintpar - BAE Integerparameter setzen (STD)

Synopsis

```
int bae_setintpar(              // Status
      int [0,[;                 // Parametertyp/-nummer:
                                //    0 = Koordinateneingaben Bereichsprüfung:
                                //        0 = Bereichsprüfung aktivieren
                                //        1 = Bereichsprüfung deaktivieren
                                //    1 = Modulwechsel Autosavemodus:
                                //        0 = Autosave ohne Benutzerabfrage
                                //        1 = Benutzerabfrage vor Autosave
                                //    2 = Anzeigedeaktivierungsmodus:
                                //        0 = Anzeige aktiviert Prüfung
                                //        1 = Anzeige deaktivieren
                                //  [ 3 = Benutzeroberfläche Menü-/Mausmodus: ]
                                //  [     Systemparameter schreibgeschützt ]
                                //    4 = Arbeitsbereichstext Farbauswahl:
                                //        0 = Standardfarben
                                //        1 = invertierte Standardfarben
                                //        2 = arbeitsbereichsspezifische Farben
                                //    5 = Element Laden Anzeigemodus:
                                //        0 = Übersichtsanzeige nach Laden
                                //        1 = Anzeige aktiviert durch bae_load
                                //    6 = Dateiauswahl Dialogmodus:
                                //        0 = alte BAE-Dateiauswahl
                                //        1 = Explorer Standardansicht
                                //        2 = Explorer Listenansicht
                                //        3 = Explorer Detailansicht
                                //        4 = Exploreransicht kleine Piktogramme
                                //        5 = Exploreransicht große Piktogramme
                                //        6 = Standard-Stil und Standard-Größe
                                //            benutzen
                                //    7 = Elementauswahl Dialogmodus:
                                //        0 = nur Name anzeigen
                                //        1 = Name und Datum anzeigen
                                //    8 = Elementauswahl Sortierfunktion:
                                //        0|1 = Sortierung nach Name
                                //        2 = Sortierung numerisch
                                //        3 = Sortierung nach Datum
                                //    9 = Anzeigemodus Platzierung:
                                //        0 = platzierte Elemente sichtbar
                                //        1 = unplatzierte Elemente sichtbar
                                //  [ 10 = Letzter Dateisystemfehler: ]
                                //  [     Systemparameter schreibgeschützt ]
                                //    11 = Kommandohistorie Modus:
                                //        0 = Kommandohistorie aktiviert
                                //        1 = Kommandohistorie deaktiviert
                                //    12 = Popupmenü Mauswarpmodus:
                                //        0 = Kein Popupmenü Mauswarp
                                //        1 = Popupmenü Mauswarp erstes Element
                                //        2 = Popupmenü Mauswarp vorselektiertes
                                //            Element
                                //        +4 = Mauspositionsrestore-Warp
                                //        +8 = Elementpickpositions-Warp
                                //    13 = Sicherungsmodus:
                                //        0 = Sicherung aktiviert
                                //        1 = Sicherung deaktiviert
                                //    14 = Minimalgröße Mausrechteck:
                                //        ]0,[ = minimale Mausrechteckgröße
                                //    15 = Anzeigemodus Mausinfo:
                                //        0 = keine kontinuierliche Infoanzeige
                                //        1 = kontinuierliche Infoanzeige
                                //    16 = Nächste Dialogbox-Identifikationsnummer
                                //    17 = Zuletzt erzeugte
                                //          Tooltipidentifikationsnummer
                                //    18 = Polygonverwurfsanzahl
```

```
                                //    19 = Polygonprüfungsabschaltung:
                                //        0 = Polygonprüfung aktivieren
                                //        1 = Polygonprüfung abschalten
                                //    20 = Kursortastenrastermodus:
                                //        0 = Eingaberaster
                                //        1 = Pixelraster
                                //    21 = Flag - Element nicht gesichert
                                //    22 = Elementbatchlademodus:
                                //        0 = keinen Batch laden
                                //        1 = Batch laden
                                //        2 = Batch laden, Zoomfenster restaurieren
                                //    23 = Rasterlinienanzeige:
                                //        0 = Punktraster
                                //        1 = Linienraster
                                //    24 = Flag - Spiegelungsanzeige
                                //    25 = Flag - Eingaberasteranzeige
                                //    26 = Mausfunktionswiederholungsmodus:
                                //        0 = Menüfunktion wiederholen
                                //        +1 = Tastenfunktion wiederholen
                                //        +2 = Kontextmenüfunktion wiederholen
                                //  [ 27 = Maximale Undo-Redo-Anzahl: ]
                                //  [     Systemparameter schreibgeschützt ]
                                //    28 = Menübaumansichtsmodus:
                                //        0 = Kein Menübaumfenster
                                //        1 = Menübaumfenster links
                                //        2 = Menübaumfenster rechts
                                //    29 = Menübaumansicht Pixelbreite
                                //    30 = Flag - Meldungshistorie deaktiviert
                                //    31 = Elementlademeldungsmodus:
                                //        0 = Standardmeldung
                                //        1 = Benutzerspezifische Meldung
                                //        2 = Benutzerspezifische Fehlermeldung
                                //    32 = Pickmarkeranzeigemodus:
                                //        0 = Kreismarker
                                //        1 = Diamantmarker
                                //    33 = Mausdrag-Status:
                                //        0 = Kein Mausdrag
                                //        1 = Mausdragangeforderung
                                //        2 = Mausdrag durchgeführt
                                //        3 = Mausdragfreigabeanforderung
                                //    34 = Flag - Menüfunktionswiederholung
                                //          angefordert
                                //    35 = Flag - Funktion abgebrochen
                                //    36 = Flag - Planauswahlvorschau
                                //    37 = Dateizugriffsfehler-Anzeigemodus:
                                //        0 = Nur Statuszeilenanzeige
                                //        1 = Bestätigungsabfrage
                                //    38 = Elementauswahl-Referenzanzeigemodus:
                                //        0 = Projektdatei-Referenzen anzeigen
                                //        1 = Bibliotheksdatei-Referenzen anzeigen
                                //    39 = Maus-Doppelklick-Modus:
                                //        0 = Doppelklick und Selektion
                                //             von 0 an rechte Maustaste zuweisen
                                //        1 = Doppelklick ignorieren
                                //        2 = Doppelklick an rechte Maustaste
                                //             zuweisen
                                //    40 = Mauspick-Doppelklick-Modus:
                                //        0 = Doppelklick an rechte Maustaste
                                //             zuweisen
                                //        1 = Doppelklick ignorieren
                                //  [ 41 = Flags - Dialogelementunterstützung: ]
                                //  [     Systemparameter schreibgeschützt ]
                                //    42 = Fortschrittsanzeigemodus:
                                //        0 = Keine Fortschrittsanzeige
                                //        1 = Fortschrittsanzeige
                                //    43 = Abbruchanforderungsflag für
                                //          Fortschrittsanzeige:
                                //    44 = Flag - Mittlere Maustaste deaktiviert:
```

```
                              //  [ 45 = Anzahl aktueller Undo-Elemente: ]
                              //  [      Systemparameter schreibgeschützt ]
                              //  [ 46 = Flag - Datei Drag-und-Drop Operation: ]
                              //  [      Systemparameter schreibgeschützt ]
                              //    47 = Flag - Automatische
                              //           BAE-Fensteraktivierung
                              //  [ 48 = Anzahl aktive Menüfunktionen ]
                              //  [      Systemparameter schreibgeschützt ]
                              //  [ 49 = Anzahl Punkte in interner Polygonliste ]
                              //  [      Systemparameter schreibgeschützt ]
                              //    50 = Priorit&auuml;t alternative
                              //           Konfigurationsdatei
                              //    51 = Sicherungsmodus für Dialogposition:
                              //        0 = Absolutkoordinaten specihern
                              //        1 = Hauptfenster-Relativkoordinaten
                              //             speichern
                              //        2 = Hauptfenster-Monitorabsolutkoordinaten
                              //             speichern
                              //    52 = Message-Box Default-Button-Index:
                              //        (-1) = Kein Default (Abbruch oder No
                              //        0-2 = Default-Button-Index
      int;                    // Parameterwert
      );
```

Beschreibung

Die Funktion **bae_setintpar** dient dazu, Systemparameter vom Typ `int` im **Bartels AutoEngineer** zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **bae_setintpar** gesetzten Systemparametern können mit der Funktion **bae_getintpar** abgefragt werden.

Siehe auch

Funktionen **bae_getdblpar**, **bae_getintpar**, **bae_getstrpar**, **bae_setdblpar**, **bae_setstrpar**.

bae_setmoduleid - BAE Modulbezeichnung setzen (STD)

Synopsis

```
int bae_setmoduleid(          // Rückgabe Status
      string;                 // Modulbezeichnung
      );
```

Beschreibung

Die Funktion **bae_setmoduleid** setzt die Bezeichnung des aktuell aktiven BAE-Programmmoduls. Der Funktionsrückgabewert ist Null bei erfolgreicher Zuweisung der Modulbezeichnung oder ungleich NUll im Fehlerfall.

Siehe auch

Funktion **bae_getmoduleid**.

bae_setmousetext - BAE Mausklick-Eingabetext definieren (STD)

Synopsis

```
int bae_setmousetext(         // Status
      string;                 // Antwort-Zeichenkette
      );
```

Beschreibung

Die Funktion **bae_setmousetext** definiert die übergebene Antwort-Zeichenkette als Default-Rückgabewert bei Aktivierung einer Maustaste für einen nachfolgenden Aufruf der Funktion **bae_readtext**. Ein Aufruf der Funktion **bae_readtext** deaktiviert gleichzeitig auch wieder die zuvor mit **bae_setmousetext** definierte Möglichkeit der Mausklick-Eingabe.

Siehe auch

Funktion **bae_readtext**.

bae_setplanfname - BAE Projekdateiname setzen (STD)

Synopsis

```
int bae_setplanfname(          // Status
      string;                  // BAE-Projektdateiname
      );
```

Beschreibung

Die Funktion **bae_setplanfname** setzt den BAE-Projektdateinamen. Die Dateinamenserweiterung `.ddb` wird automatisch angefügt, wenn ein BAE-Projektdateiname ohne Dateinamenserweiterung spezifiziert wurde. Der Rückgabewert der Funktion ergibt sich zu Null, wenn der BAE-Projektdateiname erfolgreich gesetzt wurde, zu 1 bei fehlendem bzw. falschem Funktionsparameter oder zu 2, wenn durch ein geladenes Element bereits ein BAE-Projektdateiname definiert ist.

Siehe auch

Funktion **bae_planfname**.

bae_setpopdash - BAE Popup/Toolbar Parameter für gestrichelte Linien setzen (STD)

Synopsis

```
void bae_setpopdash(
      double ]0.0,[;           // Basisstrichlinie (STD2)
      double ]-0.5,0.5[;       // Relativer Strichabstand
      );
```

Beschreibung

Die Funktion **bae_setpopdash** setzt die Basislänge und den relativen Strichabstand für die Ausgabe gestrichelter Linien in Popupmenüs bzw. Toolbars.

Siehe auch

Funktion **bae_popdrawpoly**.

bae_setstrpar - BAE Stringparameter setzen (STD)

Synopsis

```
int bae_setstrpar(              // Status
      int [0,[;                 // Parametertyp/-nummer:
                                //    0 = Aktueller Element-Kommentartext
                                //    1 = Aktuelle Element-Spezifikation
                                //  [ 2 = Systemparameter - kein Schreibzugriff ]
                                //  [ 3 = Systemparameter - kein Schreibzugriff ]
                                //    4 = Fadenkreuz-Infotext
                                //    5 = Tooltip text
                                //  [ 6 = Systemparameter - kein Schreibzugriff ]
                                //    7 = Menütext der aktuell aktiven Funktion
                                //    8 = Aktueller Menüitemelementtext
                                //    9 = Aktuelle benutzerspezifische
                                //          Elementlademeldung
                                //    10 = Zwischenablage-Textstring
                                //    11 = Modulaufruf nächstes Dateiargument
                                //    12 = Modulaufruf nächstes Elementargument
                                //    13 = Modulaufruf nächstes
                                //          Kommando-/Typargument
                                //    14 = Letzter Ausgabedateiname
                                //  [ 15 = Systemparameter - kein Schreibzugriff ]
                                //    16 = Werkzeugleistenschaltfläche
                                //          Zeichen/Resourceelement
                                //  [ 17 = Systemparameter - kein Schreibzugriff ]
                                //  [ 18 = Systemparameter - kein Schreibzugriff ]
                                //    19 = Alternatives Verzeichnis für
                                //          Konfigrationsdaten
                                //    20 = Spalte lokale Daten
                                //    21 = Spalt globale Daten
      string;                   // Parameterwert
      );
```

Beschreibung

Die Funktion **bae_setstrpar** dient dazu, Systemparameter vom Typ im `string` im **Bartels AutoEngineer** zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **bae_setstrpar** gesetzten Systemparametern können mit der Funktion **bae_getstrpar** abgefragt werden.

Siehe auch

Funktionen **bae_getdblpar**, **bae_getintpar**, **bae_getstrpar**, **bae_setdblpar**, **bae_setintpar**.

bae_settbsize - BAE Toolbarbereich definieren/anzeigen (STD)

Synopsis

```
void bae_settbsize(
      double [0.0,[;          // Toolbargröße
      int [0,3];              // Toolbar Anordnung/Ausrichtung:
                              //    0 = horizontal an unterer
                              //         Arbeitsbereichsbegrenzung
                              //    1 = vertikal an rechter
                              //         Arbeitsbereichsbegrenzung
                              //    2 = horizontal an oberer
                              //         Arbeitsbereichsbegrenzung
                              //    3 = vertikal an linker
                              //         Arbeitsbereichsbegrenzung
      );
```

Beschreibung

Mit der Funktion **bae_settbsize** kann eine Werkzeugleiste definiert bzw. angezeigt werden. Der erste Parameter gibt dabei in Abhängigkeit vom zweiten Parameter die gewünschte Größe des Toolbarbereichs an. Bei vertikaler Ausrichtung der Toolbar wird dieser Wert als Spaltenanzahl interpretiert, bei horizontaler Ausrichtung hingegen als Zeilenanzahl. Durch die Angabe des Wertes Null für die Toolbargröße kann eine bereits definierte Toolbar wieder ausgeblendet werden. Die Dimension des verfügbaren Grafikarbeitsbereichs zur Anzeige von Toolbars kann mit der Funktion **bae_mtpsize** ermittelt werden. Mit der Funktion **bae_tbsize** kann die tatsächliche Größe einer mit **bae_settbsize** definierten Toolbar nachträglich abgefragt werden. Die Funktion **bae_popsetarea** dient dazu, wahlweise die Toolbar oder das Standard-Popupmenü für nachfolgende Popupoperationen zu aktivieren. Mit der Funktion **bae_popclrtool** können sämtliche Anzeigen der aktuellen Toolbar gelöscht bzw. deaktiviert werden.

Siehe auch

Funktionen **bae_mtpsize**, **bae_popclrtool**, **bae_popsetarea**, **bae_tbsize**.

bae_storecmdbuf - BAE Kommando in Kommandohistorie speichern (STD)

Synopsis

```
int bae_storecmdbuf(          // Status
      int [0,[;               // Kommandospeichermodus
                              //    0 : Kommando an Historie anfügen
      string;                 // Kommando- bzw. Kommandosequenzzeichenkette
      string;                 // Kommandoanzeigetext
      );
```

Beschreibung

Die Funktion **bae_storecmdbuf** trägt das angegebene Kommando in die aktuelle Kommandohistorie ein. Über den Kommandospeichermodus wird angegeben wo das Kommando in der Historie einzufügen ist und ob bzw. wie die Kommandohistorie umzuorganisieren ist. Die Kommando- bzw. Kommandosequenzzeichenkette und der Kommandoanzeigetext (entsprechend der Titelleistenanzeige) werden über den zweiten und dritten Funktionsparameter angegeben. Der Funktionsrückgabewert ist Null wenn die Zuweisung erfolgreich war oder ungleich Null im anderen Fall.

Siehe auch

Funktion **bae_getcmdbuf**.

bae_storedistpoly - Internes BAE Distanzabfragepolygon speichern (STD)

Synopsis

```
int bae_storedistpoly(          // Status
      );
```

Beschreibung

Die Funktion **bae_storedistpoly** speichert das aktuell mit **bae_clearpoints** und **bae_storepoint** erzeugte Polygon als Distanzabfragepolygon. Anschließend kann die Funktion **bae_querydist** zur Bestimmung des Abstands eines gegebenen Punktes zum Distanzabfragepolygon verwendet werden. Das mit **bae_storedistpoly** gespeicherte Distanzabfragepolygon kann mit der Funktion **bae_cleardistpoly** wieder gelöscht werden. Der Rückgabewert der Funktion ist Null, wenn kein Fehler aufgetreten ist, (-1) wenn das interne Distanzabfragepolygon bereits definiert ist oder (-2) bei ungültigen Polygondaten.

Siehe auch

Funktionen **bae_cleardistpoly**, **bae_clearpoints**, **bae_querydist**, **bae_storepoint**.

bae_storeelem - BAE Element speichern (STD)

Synopsis

```
int bae_storeelem(              // Status
      string;                   // Dateiname
      string;                   // Elementname
      );
```

Beschreibung

Die Funktion **bae_storeelem** speichert das aktuell im **AutoEngineer** geladene Element unter dem angegebenen Namen in die Zieldatei. Diese Funktion entspricht der Menüfunktion Ablegen auf Namen, d.h. falls in eine andere Datei geschrieben wird, muss gegebenenfalls Update Bibliothek aufgerufen werden. Der Rückgabewert ist Null, wenn keine Fehler aufgetreten sind, (-1) bei Dateizugriffsfehlern, 1 bei ungültigen Parametern und 2 wenn kein Element im Speicher vorhanden ist.

bae_storekeyiact - BAE Tasteneingabe vorgeben (STD)

Synopsis

```
void bae_storekeyiact(
      int [0,3];                // Automatischer Interaktionsmodus (STD21)
      int;                      // Tastenzeichencode (ASCII)
      );
```

Beschreibung

Die Funktion **bae_storekeyiact** ermöglicht die Steuerung von Tasteneingaben bei der Ausführung von Menüfunktionen über die Funktion **bae_callmenu**. Ist der Interaktionsmodus auf Null gesetzt, so wird die Tasteneingabe vom Benutzer erwartet, ansonsten wird der übergebene Tastencode als Eingabe übernommen. Die Vorgaben werden in einer Queue gespeichert, d.h. es sind mehrere Vorgaben zur Steuerung ganzer Eingabeabläufe möglich.

Siehe auch

Funktionen **bae_peekiact**, **bae_storemenuiact**, **bae_storemouseiact**, **bae_storetextiact**.

bae_storemenuiact - BAE Menüwahl vorgeben (STD)

Synopsis

```
void bae_storemenuiact(
      int [0,3];              // Automatischer Interaktionsmodus (STD21)
      int;                    // Menüzeile (0..n-1)
      int [1,3];              // Maustaste (STD17)
      );
```

Beschreibung

Die Funktion **bae_storemenuiact** ermöglicht die Steuerung von Menüeingaben bei der Ausführung von Menüfunktionen über die Funktion **bae_callmenu**. Ist der Interaktionsmodus auf Null gesetzt, so wird die entsprechende Menüwahl vom Benutzer erwartet, ansonsten wird die übergebene Menüzeile als Menüwahl mit der entsprechenden Maustaste übernommen. Die Vorgaben werden in einer Queue gespeichert, d.h. es sind mehrere Vorgaben zur Steuerung ganzer Eingabeabläufe möglich.

Siehe auch

Funktionen **bae_peekiact**, **bae_storekeyiact**, **bae_storemouseiact**, **bae_storetextiact**.

bae_storemouseiact - BAE Mauseingabe vorgeben (STD)

Synopsis

```
void bae_storemouseiact(
      int [0,3];              // Automatischer Interaktionsmodus (STD21)
      double;                 // Maus X-Koordinate (STD2)
      double;                 // Maus Y-Koordinate (STD2)
      int [0,15];             // Mauskoordinatenmodus:
                              //    0 = angegebene Koordinaten mit Rasterfang
                              //         benutzen
                              //    1 = alte Mauskoordinaten benutzen
                              //    2 = angegebene Koordinaten rasterfrei benutzen
                              //    +4 = Mauszeiger auf angegebene Position setzen
                              //    +8 = BAE-Fenster aktivieren
      int [0,3];              // Maustastencode (STD17)
      [];                     // Tastatureingabe
      );
```

Beschreibung

Die Funktion **bae_storemouseiact** ermöglicht die Steuerung von Mauseingaben bei der Ausführung von Menüfunktionen über die Funktion **bae_callmenu**. Ist der Interaktionsmodus auf Null gesetzt, so wird die entsprechende Mauseingabe vom Benutzer erwartet, ansonsten werden die übergebenen Mausdaten als Eingabe übernommen. Bei Übergabe des Maustastencodes Null ist über den Tastatureingabeparameter ein Zeichen zu übergeben. Die Vorgaben werden in einer Queue gespeichert, d.h. es sind mehrere Vorgaben zur Steuerung ganzer Eingabeabläufe möglich.

Siehe auch

Funktionen **bae_callmenu**, **bae_peekiact**, **bae_storekeyiact**, **bae_storemenuiact**, **bae_storetextiact**.

bae_storepoint - Punkt in BAE-Punktliste eintragen (STD)

Synopsis

```
int bae_storepoint(            // Status
      double;                  // X-Koordinate (STD2)
      double;                  // Y-Koordinate (STD2)
      int [0,2];               // Punkttyp (STD15)
      );
```

Beschreibung

Die Funktion **bae_storepoint** trägt den übergebenen Punkt in die interne Punktliste ein. Der Rückgabewert ist ungleich Null, wenn ungültige Punktdaten spezifiziert wurden. Die interne Punktliste wird von den modulspezifischen Funktionen ***_storepoly** bzw. ***_storepath** oder der Funktion **bae_storedistpoly** zur Generierung von Polygonen und Leiterbahnen benötigt und kann mit der Funktion **bae_clearpoints** wieder gelöscht werden.

Siehe auch

Funktionen **bae_clearpoints**, **bae_getpolyrange**, **bae_storedistpoly**.

bae_storetextiact - BAE Texteingabe vorgeben (STD)

Synopsis

```
void bae_storetextiact(
      int [0,3];               // Automatischer Interaktionsmodus (STD21)
      string;                  // Eingabetext
      );
```

Beschreibung

Die Funktion **bae_storetextiact** ermöglicht die Steuerung von Texteingaben bei der Ausführung von Menüfunktionen über die Funktion **bae_callmenu**. Ist der Interaktionsmodus auf Null gesetzt, so wird die entsprechende Texteingabe vom Benutzer erwartet, ansonsten wird die übergebene Zeichenkette als Texteingabe übernommen. Die Vorgaben werden in einer Queue gespeichert, d.h. es sind mehrere Vorgaben zur Steuerung ganzer Eingabeabläufe möglich.

Siehe auch

Funktionen **bae_peekiact**, **bae_storekeyiact**, **bae_storemenuiact**, **bae_storemouseiact**.

bae_swconfig - BAE Softwarekonfiguration abfragen (STD)

Synopsis

```
int bae_swconfig(              // Software-Konfigurationscode:
                               // (-1) = ungültige Konfigurationsklasse,
                               // für Konfigurationsklasse 0:
                               //    0 = unbekanntes System
                               //    1 = Bartels ACAD-PCB
                               //    2 = BAE Professional
                               //    3 = BAE HighEnd
                               // für Konfigurationsklasse 1:
                               //    0 = nicht BAE Demo
                               //    1 = BAE Demo
                               //    2 = BAE FabView
                               // für Konfigurationsklasse 2:
                               //    ungleich Null = BAE Economy
                               // für Konfigurationsklasse 3:
                               //    0 = BAE-Standardmenü-Interface
                               //    1 = BAE-Standardmenü unter Windows
                               //    2 = BAE-Pulldownmenü unter Windows
                               //    3 = BAE-Standardmenü unter Motif
                               //    4 = BAE-Pulldownmenü unter Motif
                               // für Konfigurationsklasse 4:
                               //    ungleich Null = BAE Light
                               // für Konfigurationsklasse 5:
                               //    0 = keine BAE Schematics Software
                               //    1 = BAE Schematics
                               //    2 = BAE HighEnd Schematics
                               // für Konfigurationsklasse 6:
                               //    Versionsgenerierungsnummer (Build-Nummer)
      int;                     // Software-Konfigurationsklasse:
                               //    0 = Abfrage BAE Software-System
                               //    1 = Abfrage BAE Demo
                               //    2 = Abfrage BAE HighEnd
                               //    3 = Abfrage BAE User Interface
                               //    4 = Abfrage BAE Light
                               //    5 = Abfrage BAE Schematics
                               //    6 = Abfrage Versionsgenerierungsnummer (Build-
Nummer)
      );
```

Beschreibung

Die Funktion **bae_swconfig** ermittelt die aktuell aktivierte BAE-Softwarekonfiguration. Diese Information wird z.B. für die Funktionen **bae_defmenuprog**, **bae_callmenu** und **bae_store*iact** zur richtigen Behandlung unterschiedlicher Auswahlmenüs in **BAE Professional**, **BAE HighEnd**, **BAE Economy** und **BAE Light** mit den verschiedenen Benutzeroberflächen (BAE-Standard, Windows- bzw. Motif-Pulldownmenüs) benötigt.

Siehe auch

Funktion **bae_swversion**.

bae_swversion - BAE Softwareversion abfragen (STD)

Synopsis

```
string bae_swversion(         // Softwareversionsangabe oder
Betriebssystemeinstellung
      int;                    // Softwareversionsabfragemodus:
                              //    0 = Abfrage BAE Versionsnummer
                              //    1 = Abfrage BAE Freigabejahr (Format YY)
                              //    2 = Abfrage BAE Freigabejahr (Format YYYY)
                              //    3 = Abfrage betriebssystemspezifisches
                              //         Muster zur Erkennung beliebiger
                              //          Zeichenketten
                              //         (z.B. * unter Linux, .* unter MS-DOS)
                              //    4 = Abfrage betriebssystemspezifisches
                              //         Directorytrennzeichen
                              //         (z.B. / unter Linux, \ unter MS-DOS)
      );
```

Beschreibung

Mit der Funktion **bae_swversion** können Angaben zur BAE-Softwareversion oder betriebssystemspezifische BAE-Einstellungen entsprechend dem spezifizierten Abfragemodus abgefragt werden.

Siehe auch

Funktion **bae_swconfig**.

bae_tbsize - BAE Toolbardimensionen abfragen (STD)

Synopsis

```
void bae_tbsize(
      & double;               // Rückgabe Toolbar Textspaltenanzahl
      & double;               // Rückgabe Toolbar Textzeilenanzahl
      );
```

Beschreibung

Die Funktion **bae_tbsize** ermittelt die Größe der mit **bae_settbsize** definierten bzw. angezeigten Werkzeugleiste. Die Toolbarbreite wird im Parameter für die Textspaltenanzahl zurückgegeben, während die Toolbarhöhe im Parameter für die Textzeilenanzahl zurückgegeben wird. Mit Hilfe der Funktion **bae_charsize** können diese Werte in Standardlängeneinheiten umgerechnet werden.

Siehe auch

Funktionen **bae_charsize**, **bae_mtpsize**, **bae_settbsize**.

bae_twsize - BAE Textarbeitsbereichsgröße abfragen (STD)

Synopsis

```
void bae_twsize(
      & int;                  // Rückgabe Anzahl Spalten
      & int;                  // Rückgabe Anzahl Zeilen
      );
```

Beschreibung

Die Funktion **bae_twsize** ermittelt die aktuelle Größe des Arbeitsbereiches für Textausgaben. Die Größenangaben werden in den beiden Funktionsparametern zurückgegeben und sind als die Anzahl der Spalten bzw. Zeilen des Arbeitsbereiches zu interpretieren. Mit Hilfe dieser Informationen lassen sich Art und Form der Ausgabe auf den Textausgabebereich dynamisch an die BAE-Grafikumgebung anpassen.

Siehe auch

Funktion **bae_mtpsize**.

bae_wswinlx - BAE Arbeitsbereich linke Grenze abfragen (STD)

Synopsis

```
double bae_wswinlx(           // Koordinatenwert (STD2)
```

```
       );
```

Beschreibung

Der Rückgabewert der Funktion **bae_wswinlx** entspricht der linken Begrenzungskoordinate des aktuell sichtbaren grafischen Arbeitsbereichsfensters.

bae_wswinly - BAE Arbeitsbereich untere Grenze abfragen (STD)

Synopsis

```
double bae_wswinly(            // Koordinatenwert (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_wswinly** entspricht der unteren Begrenzungskoordinate des aktuell sichtbaren grafischen Arbeitsbereichsfensters.

bae_wswinux - BAE Arbeitsbereich rechte Grenze abfragen (STD)

Synopsis

```
double bae_wswinux(            // Koordinatenwert (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_wswinux** entspricht der rechten Begrenzungskoordinate des aktuell sichtbaren grafischen Arbeitsbereichsfensters.

bae_wswinuy - BAE Arbeitsbereich obere Grenze abfragen (STD)

Synopsis

```
double bae_wswinuy(            // Koordinatenwert (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **bae_wswinuy** entspricht der oberen Begrenzungskoordinate des aktuell sichtbaren grafischen Arbeitsbereichsfensters.

bae_wsmouse - BAE Arbeitsbereich Mausposition abfragen (STD)

Synopsis

```
void bae_wsmouse(
      & double;               // Rückgabe Maus-X-Koordinate (STD2)
      & double;               // Rückgabe Maus-Y-Koordinate (STD2)
      & int;                  // Rückgabe Mausstatus:
                              //    Bit 0 : jenseits linker
                              //             Arbeitsbereichsbegrenzung
                              //    Bit 1 : jenseits rechter
                              //             Arbeitsbereichsbegrenzung
                              //    Bit 2 : jenseits unterer
                              //             Arbeitsbereichsbegrenzung
                              //    Bit 3 : jenseits oberer
                              //             Arbeitsbereichsbegrenzung
                              //    Bit 4 : linke Maustaste gedrückt
                              //    Bit 5 : rechte Maustaste gedrückt
                              //    Bit 6 : mittlere Maustaste gedrückt
      );
```

Beschreibung

Mit der Funktion **bae_wsmouse** können die aktuellen Mauskoordinaten innerhalb des Grafikarbeitsbereichs ermittelt werden. Der Parameter zur Rückgabe des Mausstatus gibt an, welche Maustasten gerade gedrückt sind bzw. ob sich die Mausposition im Augenblick außerhalb der aktuellen Arbeitsbereichsbegrenzung befindet.

Siehe auch

Funktion **bae_popmouse**.

catext - Dateinamenserweiterung an Dateiname anhängen (STD)

Synopsis

```
void catext(
      & string;               // Dateiname
      string;                 // Dateinamenserweiterung
      );
```

Beschreibung

Die Funktion **catext** hängt die angegebene Dateinamenserweiterung an den über den ersten Funktionsparameter spezifierten Dateinamen an sofern dieser nicht bereits mit dieser Dateinamenserweiterung endet.

Siehe auch

Funktion **catextadv**.

catextadv - Dateinamenserweiterung optional an Dateiname anhängen (STD)

Synopsis

```
void catextadv(
      & string;               // Dateiname
      string;                 // Dateinamenserweiterung
      int;                    // Bearbeitungsmodus:
                              //   0 = Erweiterung nur wenn Dateiname
                              //        keine Erweiterung besitzt
                              //   1 = Annahme Dateiname besitzt keine
                              //        oder spezifizierte Erweiterung
                              //   2 = Spezifizierte Erweiterung erzwingen
      );
```

Beschreibung

Die Funktion **catextadv** hängt die angegebene Dateinamenserweiterung optional entsprechend des angegebenen Bearbeitungsmodus an den über den ersten Funktionsparameter spezifierten Dateinamen an.

Siehe auch

Funktion **catext**.

ceil - Gleitkommazahl aufrunden (STD)

Synopsis

```
double ceil(                    // Berechnungsergebnis
      double;                   // Eingabewert
      );
```

Beschreibung

Der Rückgabewert der Funktion **ceil** entspricht dem nächsten ganzzahligen Gleitkommawert, dessen Wert größer oder gleich dem des übergebenen Gleitkommawertes ist.

clock - Verbrauchte CPU-Zeit ermitteln (STD)

Synopsis

```
double clock(                   // Verbrauchte CPU-Zeit (Sekunden)
      );
```

Beschreibung

Der Rückgabewert der Funktion **clock** entspricht der seit dem Start des aktuellen BAE-Programmoduls verbrauchten CPU-Zeit in Sekunden.

con_clear - Interne Logische Netzliste löschen (STD)

Synopsis

```
void con_clear(
      );
```

Beschreibung

Die Funktion **con_clear** löscht die mit den Funktionen **con_storepart** und **con_storepin** erzeugte Logische Netzliste aus dem Hauptspeicher. Dies ist prinzipiell vor dem ersten Aufruf von **con_storepart** notwendig um evtl. vorhandene alte Netzlistendaten zu löschen. Diese Funktion sollte in jedem Fall aber auch dann aktiviert werden, wenn die im Speicher befindliche Netzliste nicht mehr benötigt wird, da andernfalls u.U. später ein Speicherüberlauf auftreten kann.

Siehe auch

Funktionen **con_storepart**, **con_storepin**, **con_write**.

con_compileloglib - Logische Bibliotheksdefinition kompilieren (STD)

Synopsis

```
int con_compileloglib(          // Status
      string;                   // DDB-Zieldateiname
      string;                   // Logische Bibliotheksdefinitionsdatei
      string;                   // Logische Bibliotheksdefinition
      );
```

Beschreibung

Die Funktion **con_compileloglib** kompiliert die angegebene(n) logischen Bibliotheksdefinitionen und speichert die kompilierten Loglib-Definitionen in der über den ersten Funktionsparameter spezifizierten DDB-Datei. Die logischen Bibliotheksdefinitionen sind in dem für das Utilityprogramm **LOGLIB** festglegten Eingabeformat bereitzustellen. Es kann entweder der Inhalt der über den zweiten Funktionsparameter spezifizierten Loglib-Definitionsdatei oder eine im dritten Funktionsparameter angegebene logische Bibliotheksdefinition kompiliert werden. Wird für einen dieser Parameter das Schlüsselwort `NULL` angegeben, dann wird der entsprechende Umsetzungsmodus deaktiviert. Der Funktionsrückgabewert ist Null bei erfolgreicher Umsetzung; im Fehlerfall wird ein Wert ungleich Null zurückgegeben.

Siehe auch

Funktionen **con_deflogpart**, **con_getlogpart**, **lay_deflibname**, **scm_defloglname**; BAE Utilityprogramm **LOGLIB**.

con_deflogpart - Logische Bauteildefinition in Bibliothek (STD)

Synopsis

```
int con_deflogpart(           // Status
      string;                 // DDB-Zieldateiname
      string;                 // Logischer Bibliotheksteilname
      string[];               // Physikalischer Bibliotheksteilname
      int [0,1];              // Flag Defaultzuweisung
      );
```

Beschreibung

Die Funktion **con_deflogpart** trägt die übergebene logische Bauteildefinition in die angegebene DDB-Zieldatei ein. Die hiermit definierte Zuweisung eines physikalischen Bibliotheksteils an ein Logikbauteil wird vom **Packager** des **AutoEngineers** zur Umsetzung logischer in physikalische Netzlisten benötigt. Der mit **con_deflogpart** definierte Bibliothekseintrag stellt eine 1:1-Zuweisung dar, d.h. der **Packager** wird bei einem derartig definierten Bauteil keine Pinnamensumsetzung vornehmen. Auch können mit **con_deflogpart** keine Gatterzuweisungen, Pin/Gate-Swaps, Stromversorgungsanschlüsse oder Attributwertzuweisungen definiert werden. Ist für den physikalischen Bibliotheksteilnamen ein Leerstring angegeben, dann erfolgt die Zuweisung an ein virtuelles Bauteil. Wird eine Liste von Namen für das physikalischen Bibliotheksteil übergeben, dann werden entsprechend dieser Namensliste zulässige Alternativbauformen für die Funktionen zum Gehäusewechsel im Layout eingetragen. Ist das Flag für die Defaultzuweisung gesetzt, dann ist eine explizite Gehäusezuweisung über das Attribut **`$plname`** erlaubt. Der Funktionsrückgabewert ist Null bei erfolgreicher Bauteildefinition, (-1) bei fehlenden bzw. ungültigen Parametern, oder (-2) wenn ein Fehler bei der Bauteildefinition aufgetreten ist.

Siehe auch

Funktionen **con_compileloglib**, **con_getlogpart**; BAE-Utilityprogramm **LOGLIB**.

con_getddbpattrib - Bauteil-/Pinattribut in DDB-Datei abfragen (STD)

Synopsis

```
int con_getddbpattrib(        // Status
      string;                 // DDB-Dateiname
      string;                 // Bauteilname
      string;                 // Pinname
      & string;               // Attributname
                              //   oder Leerstring für Planname
                              //   oder ! für erstes Attribut
                              //   oder !$attrname für nächstes Attribut
      & string;               // Attributwert Rückgabe
      );
```

Beschreibung

Die Funktion **con_getddbpattrib** ermittelt einen Bauteil- oder Pinattributwert in einer DDB-Datei. Der Funktionsrückgabewert ist 1, wenn der Attributwert gefunden wurde, Null wenn das angegebene Attribut nicht existiert, oder (-1) bei Dateizugriffsfehlern. Wird für den Pinnamen ein Leerstring angegeben, dann erfolgt eine Bauteilattributabfrage, ansonsten wird der Pinattributwert ermittelt. Bei der Spezifikation von Platzhaltern für den Attributnamen (**`!`** bzw. **`!$attrname`**) wird der darüber adressierte Attributname im entsprechenden Funktionsparamter zurückgegeben.

Siehe auch

Funktion **con_setddbpattrib**.

con_getlogpart - Logische Bauteildefinition abfragen (STD)

Synopsis

```
int con_getlogpart(            // Status
      string;                  // LOGLIB-Dateiname
      string;                  // Elementname
      int;                     // Anzahl Ausgabespalten
      & string;                // Definitionsrückgabestring
      );
```

Beschreibung

Die Funktion **con_getlogpart** schreibt die logische Bauteildefinition für das angegebene Element formatiert in den Rückgabestring. Der Rückgabewert ist gleich Null, wenn die Loglib-Information erfolgreich gespeichert wurde, (-1) bei fehlenden bzw. ungültigen Parametern, (-2) wenn die Datei nicht gefunden wurde, (-3) wenn das Element nicht in der Datei gefunden wurde oder (-4) wenn die Loglib-Information nicht erfolgreich geladen werden konnte. Für den LOGLIB-Dateinamen wird üblicherweise ein Projektdateiname oder der mit **scm_defloglname** abfragbare default Layoutbibliotheksname für den **Packager**-Lauf angegeben.

Siehe auch

Funktionen **con_compileloglib**, **con_deflogpart**, **lay_deflibname**, **scm_defloglname**; BAE Utilityprogramm **LOGLIB**.

con_setddbpattrib - Bauteil-/Pinattribut in DDB-Datei setzen (STD)

Synopsis

```
int con_setddbpattrib(         // Status
      string;                  // DDB-Dateiname
      string;                  // Bauteilname
      string;                  // Pinname
      string;                  // Attributname
      string;                  // Attributwert
      );
```

Beschreibung

Mit der Funktion **con_setddbpattrib** kann ein Bauteil- oder Pin-Attributwert in einer DDB-Datei gesetzt werden. Der Rückgabewert der Funktion ist ungleich Null, wenn ein Fehler aufgetreten ist. Die Angabe des Bauteilnamens ist zwingend, das darüber spezifizierte Bauteil muss in der Netzliste der DDB-Datei definiert sein. Die Angabe des Pinnamens ist optional; wird für den Pinnamen ein Leerstring angegeben, dann wird der Attributwert als Bauteilattribut interpretiert, ansonsten erfolgt der Eintrag eines Pinattributs. Weder Bauteil- noch Pinname dürfen Großbuchstaben enthalten. Der Attributname muss mit dem Zeichen **$** beginnen und darf ebenfalls keine Großbuchstaben enthalten. Für den Attributwert darf ein Leerstring zum Zurücksetzen des Wertes angegeben werden; die maximal speicherbare Länge des Attributwert-Strings beträgt 40 Zeichen. Bei Spezifikation des konstanten Attributwerts **`PA_NILVAL`** wird das Attribut (entsprechend der Schaltfläche Kein Wert im **Schaltplaneditor**) komplett zurückgesetzt.

Siehe auch

Funktion **con_getddbpattrib**.

con_storepart - Interne Logische Netzliste Bauteil speichern (STD)

Synopsis

```
int con_storepart(            // Status
      string;                 // Logischer Bauteilname
      string;                 // Logischer Bauteilbibliotheksname
      );
```

Beschreibung

Die Funktion **con_storepart** trägt das übergebene Logische Bauteil in die Bauteilliste der aktuell im Hauptspeicher befindlichen internen Logischen Netzliste ein. Der Rückgabewert ist gleich Null, wenn das Bauteil erfolgreich gespeichert wurde, (-1) bei fehlenden bzw. ungültigen Parametern, oder (-2) wenn das Bauteil bereits in der Netzliste definiert ist. Die interne Logische Netzliste kann mit **con_write** in einer DDB-Datei abgelegt werden oder mit **con_clear** wieder aus dem Hauptspeicher gelöscht werden.

Warnung

Durch die Verwendung der **con_store***-Funktionen wird Hauptspeicher belegt. Wenn die mit diesen Funktionen erzeugten Netzlistendaten nicht mehr benötigt werden, dann sollte in jedem Fall die im Speicher befindliche Netzliste mit **con_write** oder mit **con_clear** wieder gelöscht werden, um Speicherplatzproblemen vorzubeugen.

Siehe auch

Funktionen **con_clear**, **con_storepin**, **con_write**.

con_storepin - Interne Logische Netzliste Pin speichern (STD)

Synopsis

```
int con_storepin(             // Status
      string;                 // Logischer Bauteilname
      string;                 // Logischer Bauteilpinname
      string;                 // Netzname
      );
```

Beschreibung

Die Funktion **con_storepin** trägt den übergebenen Logischen Bauteilpin in die aktuell im Hauptspeicher befindliche Logische Netzliste ein. Voraussetzung hierfür ist, dass ein Bauteil mit dem angegebenen Namen bereits vorher mit **con_storepart** definiert wurde. Ist für den Netznamen ein Leerstring angegeben, dann wird der Pin an kein Netz der Netzliste angeschlossen; im anderen Fall wird der angegebene Pin an das entsprechende Netz angeschlossen. Ist in der Netzliste noch kein Netz mit dem angegebenen Name definiert, dann wird dieses automatisch erzeugt. Der Rückgabewert ist gleich Null, wenn der Bauteilpin erfolgreich gespeichert wurde, (-1) bei fehlenden bzw. ungültigen Parametern, (-2) wenn das Bauteil noch nicht definiert ist, (-3) wenn der Pin bereits angeschlossen ist, oder (-4) wenn zu viele Netze definiert wurden.

Warnung

Durch die Verwendung der **con_store***-Funktionen wird Hauptspeicher belegt. Wenn die mit diesen Funktionen erzeugten Netzlistendaten nicht mehr benötigt werden, dann sollte in jedem Fall die im Speicher befindliche Netzliste mit **con_write** oder mit **con_clear** wieder gelöscht werden, um Speicherplatzproblemen vorzubeugen.

Siehe auch

Funktionen **con_clear**, **con_storepart**, **con_write**.

con_write - Interne Logische Netzliste auf Datei ausgeben (STD)

Synopsis

```
int con_write(                  // Status
      string;                   // DDB-Dateiname
      string;                   // Netzlisten-Elementname
      );
```

Beschreibung

Die Funktion **con_write** dient dazu, die mit den **con_store***-Funktionen erzeugte, aktuell im Hauptspeicher befindliche Logische Netzliste in der angegebenen DDB-Datei unter dem spezifizierten Elementnamen abzulegen. Darüber hinaus löscht **con_write** die interne Netzliste wieder aus dem Hauptspeicher. Der Rückgabewert ist gleich Null, wenn die Netzliste erfolgreich gespeichert wurde, (-1) bei fehlenden bzw. ungültigen Parametern, (-2) wenn keine Netzlistendaten definiert sind, oder (-3) wenn ein Fehler bei der Ausgabe auf die DDB-Datei aufgetreten ist. Das Format der durch **con_write** in der DDB-Datei abgelegten Logischen Netzliste entspricht dem Netzlistenformat, wie es durch den **Schaltplaneditor** des **Bartels AutoEngineer** erzeugt wird, d.h. die mit **con_write** generierte Netzliste kann anschließend durch einen **Packager**-Lauf in das Layoutsystem des **Bartels AutoEngineer** übernommen werden.

Warnung

Die **con_***-Funktionen stellen ein sehr mächtiges Hilfsmittel zur Übernahme von Fremdnetzlisten in den **Bartels AutoEngineer** dar. Dennoch bleibt zu beachten, dass eine unsachgerechter Anwendung dieser Funktionen unter Umständen eine inkonsistente und kaum mehr korrigierbare Vermischung von Netzlistendaten verursachen kann. Es wird daher nachdrücklich empfohlen, stets durch eine vorherige Prüfung der in der Zieldatei abgelegten Daten eine kontrollierte, d.h. konfliktfreie Netzlistenübernahme sicherzustellen.

Siehe auch

Funktionen **con_clear**, **con_storepart**, **con_storepin**.

convstring - Zeichenkette konvertieren (STD)

Synopsis

```
string convstring(              // Rückgabe konvertierte Zeichenkette
      string;                   // Eingabezeichenkette
      int;                      // Konvertierungsmodus:
                                //    0 = Dateiname ohne Namenserweiterung ermitteln
                                //    1 = Dateiname ohne Verzeichnispfad ermitteln
                                //    2 = Dateiname ohne Namenserweiterung
                                //          und Verzeichnispfad ermitteln
      );
```

Beschreibung

Die Funktion **convstring** wandelt die Eingabezeichenkette entsprechend dem angegebenen Konvertierungsmodus um und gibt das Ergebnis der Zeichenkettenumwandlung als Rückgabewert zuück.

cos - Cosinus berechnen (STD)

Synopsis

```
double cos(                     // Berechnungsergebnis
      double;                   // Winkel (STD3)
      );
```

Beschreibung

Der Rückgabewert der Funktion **cos** entspricht dem Cosinus des übergebenen Winkels. Der übergebene Gleitkommawert wird als Bogenmaßangabe interpretiert.

cosh - Hyperbolischen Cosinus berechnen (STD)

Synopsis

```
double cosh(                    // Berechnungsergebnis
      double;                   // Winkel (STD3)
      );
```

Beschreibung

Der Rückgabewert der Funktion **cosh** entspricht dem hyperbolischen Cosinus des übergebenen Winkels. Der übergebene Gleitkommawert wird als Bogenmaßangabe interpretiert.

cvtangle - Winkel in andere Einheit umwandeln (STD)

Synopsis

```
double cvtangle(                // Winkel (in Ausgabemaßeinheiten)
      double;                   // Winkel (in Eingabemaßeinheiten)
      int [0,3];                // Eingabemaßeinheit
      int [0,3];                // Ausgabemaßeinheit
      );
```

Beschreibung

Der Rückgabewert der Funktion **cvtangle** entspricht der Konvertierung des übergebenen Winkels von der Eingabe- in die Ausgabemaßeinheit. Mögliche Werte für die Maßeinheiten sind 0 für interne Winkel (Bogenmaß, STD3), 1 für Gradmaß, 2 für Bogenmaß (STD3) und 3 für Neugrad Winkel.

cvtlength - Länge in andere Einheit umwandeln (STD)

Synopsis

```
double cvtlength(               // Länge (in Ausgabemaßeinheiten)
      double;                   // Länge (in Eingabemaßeinheiten)
      int [0,4];                // Eingabemaßeinheit
      int [0,4];                // Ausgabemaßeinheit
      );
```

Beschreibung

Der Rückgabewert der Funktion **cvtlength** entspricht der Konvertierung der übergebenen Länge von der Eingabe- in die Ausgabemaßeinheit. Mögliche Werte für die Maßeinheiten sind 0 für Meter (STD2), 1 für Inch, 2 für mm, 3 für mil und 4 für Mikrometer.

ddbcheck - DDB-Element auf verfügbarkeit prüfen (STD)

Synopsis

```
int ddbcheck(                   // Rückgabe Abfragestatus
      string;                   // DDB-Dateiname
      int;                      // DDB-Datenbankklasse (STD1)
                                //    bzw. (-1) bei gültiger DDB-Dateiprüfung
      string;                   // Elementname bzw. Leerstring bei Prüfung auf
beliebige Klassenelemente
      );
```

Beschreibung

Die Funktion **ddbcheck** prüft, ob ein Element der spezifizierten Datenbankklasse mit dem angegebenen Namen in der über den Dateinamen spezifizierten DDB-Datei existiert. Wird anstelle einer gültigen Datenbankklasse der Wert (-1) übergeben, dann prüft **ddbcheck** lediglich, ob die angegebene Datei verfügbar und eine DDB-Datei ist. Wird ein Leerstring für den Elementnamen übergeben, dann prüft **ddbcheck** lediglich, ob überhaupt Elemente der spezifizierten Datenbankklasse in der angegebenen Datei existieren. Der Funktionsrückgabewert is Null, wenn das spezifizierte DDB-Objekt existiert bzw. gefunden wurde oder (-1) im anderen Fall.

Siehe auch

Funktionen **ddbclassscan**, **ddbelemrefcount**, **ddbelemrefentry**.

ddbclassid - DDB-Elementklasse Bezeichnung abfragen (STD)

Synopsis

```
string ddbclassid(            // Rückgabe DDB-Klassenbezeichnung oder Leerstring
      int ]0,[;               // DDB-Datenbankklasse (STD1)
      );
```

Beschreibung

Die Funktion **ddbclassid** ermittelt die Bezeichnung der spezifizierten Datenbankklasse und übergibt diese mit dem Funktionsrückgabewert. Bei Spezifikation unbekannter bzw. ungültiger DDB-Datenbankklassen wird ein Leerstring zurückgegeben.

ddbclassscan - DDB-Elementklasse abarbeiten (STD)

Synopsis

```
int ddbclassscan(             // Abarbeitungsstatus
      string;                 // DDB-Dateiname
      int ]0,[;               // Element Datenbankklasse (STD1)
      * int;                  // Elementname Abarbeitungsfunktion
      );
```

Beschreibung

Die Funktion **ddbclassscan** arbeitet alle Elemente der spezifizierten Datenbankklasse aus der angegebenen DDB-Datei ab. Dabei wird für jedes Element automatisch eine benutzerdefinierte Abarbeitungsfunktion aufgerufen sofern für den entsprechenden Parameter nicht das Schlüsselwort **`NULL`** eingetragen ist. Der Funktionsrückgabewert gibt entweder die Anzahl der abgearbeiteten bzw. gefundenen Elemente an, oder ergibt sich zu (-1) bei ungültigen Parameterangaben oder bei einem Fehlerstatus aus der Abarbeitungsfunktion.

Abarbeitungsfunktion

```
int callbackfunction(         // Abarbeitungsstatus
      string ename            // Elementname
     )
{
      // Verarbeitungsprogramm
      :
      return(scanstatus);
}
```

Der Rückgabewert der Abarbeitungsfunktion sollte 1 sein, wenn die Abarbeitung fortgesetzt werden soll, 0 wenn die Abarbeitung beendet werden soll, oder (-1) bei einem (benutzerdefinierten) Fehler.

Siehe auch

Funktionen **ddbcheck**, **ddbelemrefcount**, **ddbelemrefentry**.

ddbcopyelem - DDB-Dateielement kopieren (STD)

Synopsis

```
int ddbcopyelem(              // Status
      string;                 // DDB-Quelldateiname
      string;                 // DDB-Zieldateiname
      int ]0,[;               // DDB-Datenbankklasse (STD1)
      string;                 // DDB-Elementname
      int [0,1];              // Merge Source Flag:
                              //    0 = existierende Zieldateielemente
                              //         werden nicht überschrieben
                              //    1 = existierende Zieldateielemente
                              //         werden überschrieben
      );
```

Beschreibung

Die Funktion **ddbcopyelem** kopiert das angegebene DDB-Element der spezifizierten DDB-Datenbankklasse mit allen abhängigen bzw. referenzierten Elementen von der DDB-Quelldatei in die DDB-Zieldatei. Der Elementname bleibt beim Kopiervorgang erhalten, d.h. die Quell- und Zieldateien müssen unterschiedlich sein. Der letzte Funktionsparameter gibt an, ob existierende Elemente in der Zieldatei überschrieben werden dürfen (Merge Source; Quelldatei ist Master) oder nicht (Merge Destination; Zieldatei ist Master). Der Funktionsrückgabewert ergibt sich zu Null, wenn der Kopiervorgang erfolgreich war; im anderen Fall (ungültige bzw. unvollständige Parameterangaben oder DDB-Dateizugriff fehlgeschlagen) wird der Wert (-1) zurückgegeben.

Warnungen

Die Funktion **ddbcopyelem** unterliegt *nicht* dem Undo/Redo-Mechanismus, da sie auf DDB-Dateiebene arbeitet. Daher sollte diese Funktion mit größter Vorsicht benutzt werden, um ein versehentliches Überschreiben von BAE-Design- bzw. BAE-Systemdaten zu vermeiden.

Siehe auch

Funktionen **ddbdelelem**, **ddbrenameelem**.

ddbdelelem - DDB-Dateielement löschen (STD)

Synopsis

```
int ddbdelelem(               // Status
      string;                 // DDB-Dateiname
      int ]0,[;               // DDB-Datenbankklasse (STD1)
      string;                 // DDB-Elementname
      );
```

Beschreibung

Die Funktion **ddbdelelem** löscht das angegebene DDB-Element der spezifizierten DDB-Datenbankklasse aus der angegebenen DDB-Datei. Der Funktionsrückgabewert ergibt sich zu Null, wenn der Löschvorgang erfolgreich war; im anderen Fall (ungültige bzw. unvollständige Parameterangaben oder DDB-Dateizugriff fehlgeschlagen) wird der Wert (-1) zurückgegeben.

Warnungen

Die Funktion **ddbdelelem** unterliegt *nicht* dem Undo/Redo-Mechanismus, da sie auf DDB-Dateiebene arbeitet. Daher sollte diese Funktion mit größter Vorsicht benutzt werden, um ein versehentliches Löschen von BAE-Design- bzw. BAE-Systemdaten zu vermeiden.

Siehe auch

Funktionen **ddbcopyelem**, **ddbrenameelem**.

ddbelemrefcount - DDB-Dateielement Referenzanzahl abfragen (STD)

Synopsis

```
int ddbelemrefcount(            // Rückgabe Referenzanzahl oder (-1)
      string;                   // DDB-Dateiname
      int ]0,[;                 // DDB-Datenbankklasse (STD1)
      string;                   // DDB-Elementname
      );
```

Beschreibung

Die Funktion **ddbelemrefcount** ermittelt die Anzahl der Bibliothekselemente, die von dem angegebenen DDB-Element referenziert werden. Die Angabe des DDB-Elements erfolgt durch die Spezifikation von DDB-Dateiname, DDB-Datenbankklasse sowie Elementname über die entsprechenden Funktionsparameter. Der Funktionsrückgabewert gibt die Anzahl der referenzierten Bibliothekselemente an bzw. ergibt sich zu (-1), wenn das angegebene DDB-Element nicht existiert bzw. nicht verfügbar ist. Zur selektiven Ermittlung der DDB-Datenbankklassen und der Elementnamen einzelner Bibliotheksreferenzeinträge eines DDB-Elements kann die Funktion **ddbelemrefentry** benutzt werden.

Siehe auch

Funktionen **ddbcheck**, **ddbclassscan**, **ddbelemrefentry**.

ddbelemrefentry - DDB-Dateielement Referenzeintrag abfragen (STD)

Synopsis

```
int ddbelemrefentry(            // Rückgabe Abfragestatus
      string;                   // DDB-Dateiname
      int ]0,[;                 // DDB-Datenbankklasse (STD1)
      string;                   // DDB-Elementname
      int ]0,[;                 // Referenzeintrag Index
      & int ]0,[;               // Referenzeintrag DDB-Klasse (STD1)
      & string;                 // Referenzeintrag Elementname
      );
```

Beschreibung

Die Funktion **ddbelemrefentry** ermittelt die Datenbankklasse und den Elementnamen eines durch das angegebene DDB-Element referenzierten Bibliothekselements. Die Angabe des abzufragenden Referenzeintrags erfolgt durch die Spezifikation des DDB-Dateinamens, der DDB-Datenbankklasse und des Namens des DDB-Elements sowie durch die Spezifikation eines Index in die Liste der zu diesem DDB-Element definierten Bibliotheksreferenzen. Die Anzahl der Bibliotheksreferenzen des DDB-Elements kann mit der Funktion **ddbelemrefcount** ermittelt werden und bestimmt zugleich die Obergrenze für gültige Indizes in die Referenzliste an. Der Funktionsrückgabewert ergibt sich zu Null, wenn die Abfrage erfolgreich war bzw. zu (-1) im anderen Fall.

Siehe auch

Funktionen **ddbcheck**, **ddbclassscan**, **ddbelemrefcount**.

ddbgetelemcomment - DDB-Dateielement Kommentartext abfragen (STD)

Synopsis

```
int ddbgetelemcomment(          // Status
      string;                   // DDB-Dateiname
      int ]0,[,                 // DDB-Datenbankklasse (STD1)
      string;                   // DDB-Elementname
      & string;                 // Rückgabe Kommentartext
      );
```

Beschreibung

Die Funktion **ddbgetelemcomment** dient der Abfrage von mit **ddbsetelemcomment** gesetzten Kommentartexten für DDB-Dateielemente. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) bei fehlenden oder ungütigen Parametern oder wenn das DDB-Dateielement bzw. der Kommentartext nicht gefunden wurde.

Siehe auch

Funktion **ddbsetelemcomment**.

ddbgetlaypartpin - DDB-Dateielement Layoutbauteilpindaten abfragen (STD)

Synopsis

```
int ddbgetlaypartpin(          // Rückgabe Abfragestatus:
                               //    ( 0) = kein Fehler, Abfrageergebnis gültig
                               //    (-1) = DDB-Dateizugriffsfehler
                               //    (-2) = Ungültige Parameter
                               //    (-3) = Bauteilsymbol nicht gefunden
                               //    (-4) = Pin nicht definiert
      string;                  // DDB-Dateiname
      string;                  // Layoutbauteilsymbolname
      int [0,[;                // Pinindex
      & string;                // Pinname
      & string;                // Pinsymbolname
      & double;                // Pin-X-Position (STD2)
      & double;                // Pin-Y-Position (STD2)
      & double;                // Pin-Drehwinkel (STD3)
      & int;                   // Pin-Spiegelungsmodus (STD14)
      );
```

Beschreibung

Mit der Funktion **ddbgetlaypartpin** können in der angegebenen DDB-Datei Informationen über die auf dem angegebenen Layoutbauteilsymbol definierten Pins abgefragt werden. Zur Abfrage aller Pins eines existierenden Layoutbauteilsymbols kann die Funktion solange mit aufsteigendem Pinindex aufgerufen werden, bis der Funktionsrückgabewert einen nicht definierten Pin signalisiert.

Siehe auch

Funktionen **ddbcheck**, **ddbclassscan**, **ddbelemrefcount**, **ddbelemrefentry**.

ddbrenameelem - DDB-Dateielement umbenennen (STD)

Synopsis

```
int ddbrenameelem(             // Status
      string;                  // DDB-Dateiname
      int ]0,[,                // DDB-Datenbankklasse (STD1)
      string;                  // Alter DDB-Elementname
      string;                  // Neuer DDB-Elementname
      );
```

Beschreibung

Die Funktion **ddbrenameelem** ändert den Elementnamen des angegebenen DDB-Dateielements. Der Funktionsrückgabewert ergibt sich zu Null bei erfolgreicher Umbenennung; im anderen Fall (ungültige bzw. unvollständige Parameterangaben oder DDB-Dateizugriff fehlgeschlagen) wird der Wert (-1) zurückgegeben.

Warnungen

Die Funktion **ddbrenameelem** unterliegt *nicht* dem Undo/Redo-Mechanismus, da sie auf DDB-Dateiebene arbeitet. Daher sollte diese Funktion mit größter Vorsicht benutzt werden, um ein versehentliches Umbenennen von BAE-Design- bzw. BAE-Systemdaten zu vermeiden.

Siehe auch

Funktionen **ddbcopyelem**, **ddbdelelem**.

ddbsetelemcomment - DDB-Dateielement Kommentartext setzen (STD)

Synopsis

```
int ddbsetelemcomment(        // Status
      string;                 // DDB-Dateiname
      int ]0,[,               // DDB-Datenbankklasse (STD1)
      string;                 // DDB-Elementname
      string;                 // Kommentartext
      );
```

Beschreibung

Mit der Funktion **ddbsetelemcomment** können Kommentartexte zur späteren Abfrage mit der Funktion **ddbgetelemcomment** an DDB-Dateielemente zugewiesen werden. Der Funktionsrückgabewert ist Null bei erfolgreicher Zuweisung oder (-1) bei fehlenden oder ungütigen Parametern oder wenn das DDB-Dateielement bzw. der Kommentartext nicht gefunden wurde. Befindet sich das bearbeitete DDB-Element gerade im Arbeitsspeicher, dann wird die Kommentarzuweisung auch dort durchgeführt.

Siehe auch

Funktion **ddbgetelemcomment**.

ddbupdtime - DDB-Dateielement Änderungsdatum abfragen (STD)

Synopsis

```
int ddbupdtime(               // Status
      string;                 // Dateiname
      int [100,[;             // Elementklasse (STD1)
      string;                 // Elementname
      & int;                  // Rückgabe Sekunden nach Minutenanfang
      & int;                  // Rückgabe Minuten nach Stundenanfang
      & int;                  // Rückgabe Stunden nach Mitternacht
      & int;                  // Rückgabe Tage nach Monatsanfang
      & int;                  // Rückgabe Monate nach Jahresanfang
      & int;                  // Rückgabe Jahr
      );
```

Beschreibung

Die Funktion **ddbupdtime** gibt in den Rückgabeparametern Datum und Zeit der letzten Änderung, die an dem angegebenen Datanbankelement durchgeführt wurde, zurück. Der Rückgabewert ist 1 wenn keine Fehler aufgetreten sind, 0 wenn das angegebene Element nicht in der Datenbankdatei vorhanden ist und (-1) wenn beim Dateizugriff Fehler aufgetreten sind oder die übergebenen Parameter ungültig sind.

dirscan - Dateiverzeichniseinträge abarbeiten (STD)

Synopsis

```
int dirscan(                   // Abarbeitungsstatus
      string;                  // Verzeichnispfadname
      string;                  // Dateinamenserweiterung:
                               //    .EXT = Namensendung .EXT
                               //    .*   = alle Dateien/Verzeichnisse
      * int;                   // Dateiname Abarbeitungsfunktion
      );
```

Beschreibung

Die Funktion **dirscan** arbeitet alle Dateinamen mit der spezifizierten Dateinamenserweiterung aus dem angegebenen Verzeichnis ab. Dabei wird für jeden Dateinamen automatisch eine benutzerdefinierte Abarbeitungsfunktion aufgerufen sofern für den entsprechenden Parameter nicht das Schlüsselwort **NULL** eingetragen ist. Der Funktionsrückgabewert gibt entweder die Anzahl der abgearbeiteten bzw. gefundenen Dateinamen an, oder ergibt sich zu (-1) bei ungültigen Parameterangaben oder bei einem Fehlerstatus aus der Abarbeitungsfunktion.

Abarbeitungsfunktion

```
int callbackfunction(          // Abarbeitungsstatus
      string fname             // Dateiname
     )
{
      // Verarbeitungsprogramm
      :
      return(scanstatus);
}
```

Der Rückgabewert der Abarbeitungsfunktion sollte 1 sein, wenn die Abarbeitung fortgesetzt werden soll, 0 wenn die Abarbeitung beendet werden soll, oder (-1) bei einem (benutzerdefinierten) Fehler.

existddbelem - DDB-Dateielement Existenz prüfen (STD)

Synopsis

```
int existddbelem(              // Status
      string;                  // Dateiname
      int [100,[;              // Elementklasse (STD1)
      string;                  // Elementname
      );
```

Beschreibung

Der Rückgabewert der Funktion **existddbelem** gibt an, ob das angegebene Element nicht in der Datenbankdatei vorhanden ist. Der Rückgabewert ist 1 wenn das Element gefunden wurde, 0 wenn das angegebene Element nicht in der Datenbankdatei vorhanden ist und (-1) wenn beim Dateizugriff Fehler aufgetreten sind oder die übergebenen Parameter ungültig sind.

exit - Programm verlassen (STD)

Synopsis

```
void exit(
      int;                     // Rückgabe Status
      );
```

Beschreibung

Die Funktion **exit** beendet das aktuelle **User Language**-Programm mit dem übergebenen Status.

Warnung

Der Rückgabestatus wird derzeit nicht von der Interpreterumgebung ausgewertet.

Siehe auch

Funktion **ulsystem_exit**.

exp - Exponentialfunktion (STD)

Synopsis

```
double exp(                     // Berechnungsergebnis
      double;                   // Eingabewert
      );
```

Beschreibung

Der Rückgabewert der Funktion **exp** entspricht dem Wert der Exponentialfunktion für den übergebenen Gleitkommawert.

fabs - Absolutwert eines Gleitkommawertes (STD)

Synopsis

```
double fabs(                    // Berechnungsergebnis
      double;                   // Eingabewert
      );
```

Beschreibung

Der Rückgabewert der Funktion **fabs** entspricht dem Absolutwert des übergebenen Gleitkommawertes.

fclose - Datei schließen (STD)

Synopsis

```
int fclose(                     // Status
      int;                      // Dateideskriptor
      );
```

Beschreibung

Die Funktion **fclose** schließt die durch den Dateideskriptor beschriebene Datei. Der Rückgabewert ist ungleich Null, wenn beim Schließen der Datei ein Fehler aufgetreten ist.

fcloseall - Alle offenen Dateien schließen (STD)

Synopsis

```
int fcloseall(                  // Status
      );
```

Beschreibung

Die Funktion **fcloseall** schließt alle durch das aktuelle **User Language**-Programm geöffneten Dateien. Der Rückgabewert ist ungleich Null, wenn beim Schließen der Dateien ein Fehler aufgetreten ist.

feof - Prüfen ob Dateiende erreicht (STD)

Synopsis

```
int feof(                       // Status
      int;                      // Dateideskriptor
      );
```

Beschreibung

Der Rückgabewert der Funktion **feof** ist ungleich Null, wenn das Ende der durch den Dateideskriptor beschriebenen Datei erreicht ist.

fgetc - Zeichen aus Datei einlesen (STD)

Synopsis

```
int fgetc(                        // Zeichencode (oder -1 bei EOF)
      int;                        // Dateideskriptor
      );
```

Beschreibung

Die Funktion **fgetc** liest ein Zeichen aus der durch den Dateideskriptor beschriebenen Datei. Der Rückgabewert der Funktion entspricht dem gelesenen Zeichen oder (-1), wenn beim Lesen ein Fehler aufgetreten ist, oder wenn das Dateiende erreicht ist. Bei einem Rückgabewert von (-1) sollte also in jedem Fall (mit der Funktion **feof**) überprüft werden, ob das Dateiende erreicht ist. Hierzu ist vorher der Fehlerbehandlungsmodus für die Dateizugriffsfunktionen entsprechend zu setzen (siehe Funktion **fseterrmode**).

fgets - Zeichenkette aus Datei einlesen (STD)

Synopsis

```
int fgets(                        // Status
      & string;                   // Rückgabe Zeichenkette
      int;                        // Maximale Einleselänge
      int;                        // Dateideskriptor
      );
```

Beschreibung

Die Funktion **fgets** liest eine Zeichenkette aus der durch den Dateideskriptor beschriebenen Datei. Der Rückgabewert der Funktion ist ungleich Null, wenn beim Lesen ein Fehler aufgetreten ist, oder wenn das Dateiende erreicht ist. Bei einem Rückgabewert von (-1) sollte also in jedem Fall (mit der Funktion **feof**) überprüft werden, ob das Dateiende erreicht ist. Hierzu ist vorher der Fehlerbehandlungsmodus für die Dateizugriffsfunktionen entsprechend zu setzen (siehe Funktion **fseterrmode**).

filemode - Dateimodus abfragen (STD)

Synopsis

```
int filemode(                     // Dateimodus
      string;                     // Dateiname
      );
```

Beschreibung

Die Funktion **filemode** ermittelt den Dateimodus bzw. die Zugriffsberechtigung für die angegebene Datei. Der Funktionsrückgabewert ist 0 bei Schreibzugriff, 1 bei Lesezugriff oder (-1), wenn der Zugriff auf die Datei fehlgeschlagen ist.

Siehe auch

Funktionen **filesize**, **filetype**.

filesize - Dateigröße abfragen (STD)

Synopsis

```
int filesize(                     // Dateigröße in Bytes oder (-1) bei Fehler
      string;                     // Dateiname
      );
```

Beschreibung

Die Funktion **filesize** ermittelt die Dateigröße der angegebenen Datei. Der Funktionsrückgabewert gibt entweder die ermittelte Dateigröße in Bytes an oder ergibt sich zu (-1), wenn der Zugriff auf die Datei fehlgeschlagen ist.

Siehe auch

Funktionen **filemode**, **filetype**.

filetype - Dateityp abfragen (STD)

Synopsis

```
int filetype(                 // Rückgabe Dateityp:
                              //    (-1) = Dateizugriff fehlgeschlagen
                              //    ( 0) = Verzeichnis
                              //    ( 1) = Reguläre Datei
                              //    ( 2) = Textdatei
      string;                 // Dateiname
      );
```

Beschreibung

Die Funktion **filetype** ermittelt den Typ der angegebenen Datei. Der Funktionsrückgabewert gibt entweder den ermittelten Dateityp an oder ergibt sich zu (-1), wenn der Zugriff auf die Datei fehlgeschlagen ist.

Siehe auch

Funktionen **filemode**, **filesize**.

floor - Gleitkommawert abrunden (STD)

Synopsis

```
double floor(                 // Berechnungsergebnis
      double;                 // Eingabewert
      );
```

Beschreibung

Der Rückgabewert der Funktion **floor** entspricht dem nächsten ganzzahligen Gleitkommawert, dessen Wert kleiner oder gleich dem des übergebenen Gleitkommawertes ist.

fmod - Gleitkommadivision Rest berechnen (STD)

Synopsis

```
double fmod(                  // Berechnungsergebnis
      double;                 // Dividend
      double;                 // Divisor
      );
```

Beschreibung

Der Rückgabewert der Funktion **fmod** entspricht dem Rest der Division von Dividend und Divisor.

fopen - Datei öffnen (STD)

Synopsis

```
int fopen(                      // Status
      string;                   // Dateiname
      int [0,14];               // Zugriffsmodus:
                                //    0 = r Lesen (einzig gültiger
                                //         BAE Demo Zugriffsmodues ohne |8)
                                //    1 = w Schreiben
                                //    2 = a Anfügen
                                //    3 = rb Lesen binär
                                //    4 = wb Schreiben binär
                                //    5 = ab Anfügen binär
                                //    |8 = Datei automatisch schließen
                                //          (für BAE Demo Schreibtests)
      );
```

Beschreibung

Die Funktion **fopen** öffnet die angegebene Datei zur Bearbeitung mit dem gewünschten Zugriffsmodus. Bei Schreibzugriff wird die Datei gegebenenfalls neu erzeugt. Der Rückgabewert der Funktion beträgt (-1), wenn beim Öffnen/Erzeugen der Datei ein Fehler aufgetreten ist, oder entspricht dem zur weiteren Bearbeitung benötigten Dateideskriptor.

Einschränkung

In den Demo-Konfigurationen der BAE-Software kann mit **fopen** nur lesend auf Dateien zugegriffen werden.

fprintf - Formatierte Ausgabe auf Datei (STD)

Synopsis

```
int fprintf(                    // Status
      int;                      // Dateideskriptor
      string;                   // Formatzeichenkette
      []                        // Parameterliste
      );
```

Beschreibung

Die Funktion **fprintf** schreibt die in der Parameterliste enthaltenen Daten formatiert in die durch den Dateideskriptor beschriebene Datei. Die Formatzeichenkette enthält Informationen, wie die Formatierung stattzufinden hat. Der Rückgabewert der Funktion ist ungleich Null, wenn beim Schreiben ein Fehler aufgetreten ist.

Formatzeichenkette

Mit Hilfe der Formatzeichenkette wandelt die Funktion nachfolgende Parameterwerte um und gibt sie entsprechend formatiert aus. Die Formatzeichenkette kann gewöhnliche Zeichen und Formatelemente enthalten. Gewöhnliche Zeichen werden unverändert ausgegeben, während Formatelemente jeweils die Umwandlung und formatierte Ausgabe des nächstfolgenden Parameterwertes veranlassen. Jedes Formatelement beginnt mit dem Prozentzeichen `%` und wird durch ein Formatkontrollzeichen abgeschlossen. Gültige Formatkontrollzeichen sind:

Zeichen	Ausgabedatentyp
`d`	Dezimal-Darstellung
`o`	Oktal-Darstellung
`x`	Hexadezimal-Darstellung (Kleinschreibung)
`X`	Hexadezimal-Darstellung (Großschreibung)
`u`	Vorzeichenlose Dezimal-Darstellung
`c`	Zeichen
`s`	Zeichenkette
`e`	Fließkomma-Darstellung (Kleinschreibung)
`E`	Fließkomma-Darstellung (Großschreibung)
`f`	Festkomma-Darstellung
`g`	`e` oder `f`, jeweils kürzere Darstellungsform
`G`	`E` oder `f`, jeweils kürzere Darstellungsform
`%`	Ausgabe des Prozentzeichens `%`

Zwischen dem Prozentzeichen und dem Formatkontrollzeichen können der Reihe nach noch folgende Angaben eingetragen sein (n = Zahlenwert):

Zeichen	Ausgabeformat
`-`	Linksbündigkeit (default: rechtsbündig)
`+`	Vorzeichenausgabe auch bei positivem numerischen Wert
`SPACE`	Ausgabe eines Leerzeichens bei positivem Wert
`#`	oktale Ausgabe mit führender `0` bzw. hexadezimale Ausgabe mit führendem `0x` oder `0X` bzw. Ausgabe mit Dezimalpunkt bei `e`, `E`, `f`, `g`, `G` bzw. Nachkommanullen bei `g`, `G`
`0`	Ausgabe führender Nullen bei numerischen Werten
`n`	Feldlänge; d.h. minimale Ausgabelänge
`.n`	Genauigkeit; d.h. Anzahl anzuzeigender Zeichen (bei `s`) bzw. Anzahl Nachkommastellen (bei `f`, `g`, `G`)
`l`	long int Dezimal-Darstellung (bei `d`, `o`, `X`, `X`)

Bei `e`, `E`, `f`, `g` und `G` werden per Default sechs Nachkommastellen ausgegeben. Ist anstelle der numerischen Angabe für die Feldlänge bzw. für die Genauigkeit das Zeichen `*` angegeben, dann wird der entsprechende Wert aus dem nächsten noch nicht abgearbeiteten Funktionsparameter übernommen. Die Ausgabe eines Prozentzeichens kann durch die Angabe `%%` veranlasst werden. Durch Backslash `\` gekennzeichnete Steuerzeichen werden wie gewöhnliche Zeichen behandelt.

Warnung

Die Anzahl und der Typ der dem Formatstring nachfolgenden Parameterwerte muss mit der Anzahl der Paare der Prozent- und Formatkontrollzeichen im Formatstring übereinstimmen. Andernfalls wird "Speichermüll" ausgegeben.

Siehe auch

Funktionen **printf**, **sprintf**.

fputc - Zeichen in Datei schreiben (STD)

Synopsis

```
int fputc(                    // Status
      char;                   // Zeichen
      int;                    // Dateideskriptor
      );
```

Beschreibung

Die Funktion **fputc** schreibt das übergebene Zeichen in die durch den Dateideskriptor beschriebene Datei. Der Rückgabewert der Funktion ist ungleich Null, wenn beim Schreiben ein Fehler aufgetreten ist.

fputs - Zeichenkette in Datei schreiben (STD)

Synopsis

```
int fputs(                    // Status
      string;                 // Zeichenkette
      int;                    // Dateideskriptor
      );
```

Beschreibung

Die Funktion **fputs** schreibt die übergebene Zeichenkette in die durch den Dateideskriptor beschriebene Datei. Der Rückgabewert der Funktion ist ungleich Null, wenn beim Schreiben ein Fehler aufgetreten ist.

frexp - Exponentialdarstellung ermitteln (STD)

Synopsis

```
double frexp(                 // Berechnungsergebnis
      double;                 // Eingabewert
      & int;                  // Exponentrückgabe
      );
```

Beschreibung

Der Rückgabewert der Funktion **frexp** ist die Mantisse der Exponentialdarstellung der übergebenen Gleitkommazahl. Der Exponent wird über Parameter zurückgegeben.

fseterrmode - Dateifehler-Behandlungsmodus setzen (STD)

Synopsis

```
int fseterrmode(              // Status
      int [0,1];              // Fehlermodus
      );
```

Beschreibung

Die Funktion **fseterrmode** setzt den Fehlerbehandlungsmodus für Dateizugriffsfehler. Wenn der Modus auf eins gesetzt ist, wird bei Dateizugriffsfehlern das **User Language**-Programm von der aktuellen Interpreterumgebung mit einer entsprechenden Fehlermeldung abgebrochen. Bei einem Modus von Null geben die Dateizugriffsfunktionen einen entsprechenden Fehlercode zurück; die Fehlerbehandlung ist in diesem Fall durch das **User Language**-Programm selbst durchzuführen. Die Defaulteinstellung für den Fehlerbehandlungsmodus ist eins.

Warnung

Bei der Verwendung von Dateizugriffsfunktionen, deren Rückgabestatus entweder als Dateizugriffs-Fehler oder z.B. als Dateiende interpretiert werden kann (siehe Funktionen **fgetc**, **fgets**), sollte in jedem Fall der Fehlerbehandlungsmodus auf Null gesetzt werden, um zu verhindern, dass die Interpreterumgebung das **User Language**-Programm beim Erreichen des Dateiendes mit einem Fehler abbricht.

get_date - Systemdatum ermitteln (STD)

Synopsis

```
void get_date(
      & int;                  // Tag (1..31)
      & int;                  // Monat (0..11)
      & int;                  // Jahre seit 1900
      );
```

Beschreibung

Die Funktion **get_date** gibt in den Parametern das aktuelle Systemdatum zurück.

get_time - Systemuhrzeit ermitteln (STD)

Synopsis

```
void get_time(
      & int;                  // Stunden seit Mitternacht (0..23)
      & int;                  // Minuten seit Stundenanfang (0..59)
      & int;                  // Sekunden seit Minutenanfang (0..59)
      );
```

Beschreibung

Die Funktion **get_time** gibt in den Parametern die aktuelle Systemuhrzeit zurück.

getchr - Zeichen einlesen (STD)

Synopsis

```
char getchr(                    // Zeichen
      );
```

Beschreibung

Die Funktion **getchr** liest ein Zeichen von der Tastatur ein. Der Rückgabewert der Funktion entspricht dem gelesenen Zeichen.

getcwd - Pfadname des Arbeitsverzeichnisses abfragen (STD)

Synopsis

```
string getcwd(                  // Rückgabe Pfadname
      );
```

Beschreibung

Die Funktion **getcwd** ermittelt den Pfadnamen des Arbeitsverzeichnisses übergibt diesen als Funktionsrückgabewert.

getenv - Umgebungsvariable abfragen (STD)

Synopsis

```
int getenv(                     // Status
      string;                   // Variablenname
      & string;                 // Variablenwert
      );
```

Beschreibung

Die Funktion **getenv** durchsucht die Liste der Betriebssystemumgebungsvariablen nach dem angegebenen Variablennamen und übergibt den zugehörigen Variablenwert im entsprechenden Parameter. Der Funktionsrückgabewert ist Null, wenn die Variable gefunden wurde bzw. ungleich Null, wenn keine Umgebungsvariable mit dem angegebenen Namen definiert ist (in diesem Fall bleibt der Parameter für den Variablenwert unverändert).

Siehe auch

Funktion **putenv**.

getextprog - Dateitypspezifische Applikation ermitteln (STD)

Synopsis

```
int getextprog(                 // Status
      string;                   // Dateinamenserweiterung
      & string;                 // Kommandozeichenkette
      );
```

Beschreibung

Die Funktion **getextprog** ermittelt die Applikation bzw. das Kommando zum Öffnen des durch die angegene Dateinamenserweiterung spezifizierten Dateityps. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) wenn keine Applikation für die angegebene Dateinamenserweiterung ermittelt werden konnte.

getstr - Zeichenkette einlesen (STD)

Synopsis

```
int getstr(                    // Status
      & string;                // Rückgabe Zeichenkette
      int;                     // Maximale Zeichenkettenlänge
      );
```

Beschreibung

Die Funktion **getstr** liest Zeichen von der Tastatur in die übergebene Zeichenkette ein, bis die Return- bzw. Eingabetaste gedrückt wird oder die maximale Anzahl Zeichen erreicht ist. Der Rückgabewert ist ungleich Null, wenn ungültige Parameter übergeben wurden.

isalnum - Prüfen ob Zeichen alphanumerisch (STD)

Synopsis

```
int isalnum(                   // Bool'sches Prüfergebnis
      char;                    // Eingabezeichen
      );
```

Beschreibung

Der Rückgabewert der Funktion **isalnum** gibt an, ob es sich bei dem übergebenen Zeichen um ein alphanumerisches Zeichen (Buchstabe oder Ziffer) handelt. Der Wert ist ungleich Null, wenn die Bedingung erfüllt ist und Null wenn nicht.

isalpha - Prüfen ob Zeichen Buchstabe (STD)

Synopsis

```
int isalpha(                   // Bool'sches Prüfergebnis
      char;                    // Eingabezeichen
      );
```

Beschreibung

Der Rückgabewert der Funktion **isalpha** gibt an, ob es sich bei dem übergebenen Zeichen um einen Buchstaben handelt. Der Wert ist ungleich Null, wenn die Bedingung erfüllt ist und Null wenn nicht.

iscntrl - Prüfen ob Zeichen Kontrollzeichen (STD)

Synopsis

```
int iscntrl(                   // Bool'sches Prüfergebnis
      char;                    // Eingabezeichen
      );
```

Beschreibung

Der Rückgabewert der Funktion **iscntrl** gibt an, ob es sich bei dem übergebenen Zeichen um ein Kontrollzeichen handelt. Der Wert ist ungleich Null, wenn die Bedingung erfüllt ist und Null wenn nicht.

isdigit - Prüfen ob Zeichen Ziffer (STD)

Synopsis

```
int isdigit(                   // Bool'sches Prüfergebnis
      char;                    // Eingabezeichen
      );
```

Beschreibung

Der Rückgabewert der Funktion **isdigit** gibt an, ob es sich bei dem übergebenen Zeichen um eine Ziffer handelt. Der Wert ist ungleich Null, wenn die Bedingung erfüllt ist und Null wenn nicht.

isgraph - Prüfen ob Zeichen sichtbares Zeichen (STD)

Synopsis

```
int isgraph(                    // Bool'sches Prüfergebnis
      char;                     // Eingabezeichen
      );
```

Beschreibung

Der Rückgabewert der Funktion **isgraph** gibt an, ob es sich bei dem übergebenen Zeichen um ein Zeichen mit sichtbarer Bilddarstellung handelt. Der Wert ist ungleich Null, wenn die Bedingung erfüllt ist und Null wenn nicht.

islower - Prüfen ob Zeichen Kleinbuchstabe (STD)

Synopsis

```
int islower(                    // Bool'sches Prüfergebnis
      char;                     // Eingabezeichen
      );
```

Beschreibung

Der Rückgabewert der Funktion **islower** gibt an, ob es sich bei dem übergebenen Zeichen um einen Kleinbuchstaben handelt. Der Wert ist ungleich Null, wenn die Bedingung erfüllt ist und Null wenn nicht.

isprint - Prüfen ob Zeichen druckbares Zeichen (STD)

Synopsis

```
int isprint(                    // Bool'sches Prüfergebnis
      char;                     // Eingabezeichen
      );
```

Beschreibung

Der Rückgabewert der Funktion **isprint** gibt an, ob es sich bei dem übergebenen Zeichen um ein druckbares Zeichen handelt. Der Wert ist ungleich Null, wenn die Bedingung erfüllt ist und Null wenn nicht.

ispunct - Prüfen ob Zeichen Satzzeichen (STD)

Synopsis

```
int ispunct(                    // Bool'sches Prüfergebnis
      char;                     // Eingabezeichen
      );
```

Beschreibung

Der Rückgabewert der Funktion **ispunct** gibt an, ob es sich bei dem übergebenen Zeichen um ein Satzzeichen handelt. Der Wert ist ungleich Null, wenn die Bedingung erfüllt ist und Null wenn nicht.

isspace - Prüfen ob Zeichen Zwischenraumzeichen (STD)

Synopsis

```
int isspace(                    // Bool'sches Prüfergebnis
      char;                     // Eingabezeichen
      );
```

Beschreibung

Der Rückgabewert der Funktion **isspace** gibt an, ob es sich bei dem übergebenen Zeichen um ein Zwischenraumzeichen handelt. Der Wert ist ungleich Null, wenn die Bedingung erfüllt ist und Null wenn nicht.

isupper - Prüfen ob Zeichen Großbuchstabe (STD)

Synopsis

```
int isupper(                    // Bool'sches Prüfergebnis
      char;                     // Eingabezeichen
      );
```

Beschreibung

Der Rückgabewert der Funktion **isupper** gibt an, ob es sich bei dem übergebenen Zeichen um einen Großbuchstaben handelt. Der Wert ist ungleich Null, wenn die Bedingung erfüllt ist und Null wenn nicht.

isxdigit - Prüfen ob Zeichen Hex-Ziffer (STD)

Synopsis

```
int isxdigit(                   // Bool'sches Prüfergebnis
      char;                     // Eingabezeichen
      );
```

Beschreibung

Der Rückgabewert der Funktion **isxdigit** gibt an, ob es sich bei dem übergebenen Zeichen um eine Hexadezimalziffer handelt. Der Wert ist ungleich Null, wenn die Bedingung erfüllt ist und Null wenn nicht.

kbhit - Prüfen ob Taste betätigt (STD)

Synopsis

```
int kbhit(                      // Tastaturstatus
      );
```

Beschreibung

Der Rückgabewert der Funktion **kbhit** gibt den aktuellen Status der Tastatureingabe an. Ein Rückgabewert von 0 bedeutet es wurde keine Taste gedrückt. Eine 1 bedeutet, dass ein Taste gedrückt wurde. Der Tastencode verbleibt dabei im Tastaturpuffer und kann mit **getchr** oder anderen Tastatureingabefunktionen eingelesen werden.

Siehe auch

Funktion **kbstate**.

kbstate - Umschalt-/Steuerungs-Tastenstatus abfragen (STD)

Synopsis

```
int kbstate(                    // Tastaturstatus (Bitwerte):
                                //    0x***1 = Umschalttaste gedrückt
                                //    0x***2 = Steuerungstaste gedrückt
                                //    0x**1* = Linke Alt-Taste gedrückt
                                //    0x**2* = Rechte Alt-Taste gedrückt
      );
```

Beschreibung

Mit der Funktion **kbstate** kann abgefragt werden, ob die Umschalttaste `Shift`, die Steuerungstaste `Ctrl` bzw. die `Alt`-Tasten gerade gedrückt sind.

Siehe auch
Funktion **kbhit**.

launch - Betriebssystemkommando absetzen ohne die Ausführung abzuwarten (STD)

Synopsis

```
int launch(                    // Status
      string;                  // Kommando
      );
```

Beschreibung

Die Funktion **launch** aktiviert das als Zeichenkette übergebene Kommando auf Betriebssystemebene. Das übergebene Kommando wird als Aufruf zum Start bzw. zur Ausführung einer Applikation bzw. eines Programms interpretiert, und die Ablaufkontrolle wird unmittelbar nach Absetzen des Kommandos wieder an BAE zurückgegeben (die aktivierte Applikation läuft dann unabhängig von BAE). Der Rückgabewert der Funktion ergibt sich zu Null, wenn das Kommando erfolgreich abgesetzt wurde; andernfalls wird ein Wert ungleich Null zurückgegeben.

Einschränkungen

Die Funktion **launch** kann nicht in der **BAE Demo**-Software angewendet werden.

Unter MS-DOS benötigt der **Phar Lap 386|DOS Extender** ausreichend konventionellen Speicher zur Ausführung von (Child-)Prozessen. Die Größe des zur Verfügung stehenden konventionellen Speichers wird mit den Optionen **`-MINREAL`** und **`-MAXREAL`** des **Phar Lap 386|DOS Extender**s festgelegt. Zur Ausführung von **User Language**-Programmen, die die **launch**-Funktion benutzen, sind die **User Language**-Interpreterumgebungen mit Hilfe des mit der BAE-Software ausgelieferten CFIG386-Tools von Phar Lap wie folgt umzukonfigurieren:

```
>  cfig386 <EXEFILE> -maxreal 0ffffh ↵
```

Für **`<EXEFILE>`** ist jeweils der Name des **User Language Interpreters** (**`scm.exe`**, **`ged.exe`**, **`neurrut.exe`**, **`cam.exe`**, **`gerview.exe`** bzw. **`ced.exe`**) einzusetzen.

Warnungen

Beachten Sie, dass die **launch**-Funktion grundlegende Mehrprozess- bzw. Multitasking-Techniken voraussetzt, die auf PC-basierenden Systemen u.U. nicht ausreichend unterstützt werden, oder die in Rechnernetzwerken (abhängig vom auszuführenden Kommando) Probleme bereiten können.

Es wird dringend empfohlen, die Standardausgabe von DOS-Kommandos auf temporäre Dateien umzulenken und zur Anzeige eine **User Language**-Funktion zur Dateibetrachtung zu verwenden, da andernfalls die BAE-Grafikoberfläche von der Standardausgabe der mit **launch** abgesetzten DOS-Kommandos überschrieben wird.

Die Fehlerausgabe von DOS-Kommandos erfolgt grundsätzlich auf den Bildschirm, wodurch die BAE-Grafikoberfläche überschrieben wird. Da unter DOS prinzipiell keine Möglichkeit zur Umlenkung der Fehlerausgabe besteht, ist dieses Problem nur dadurch zu umgehen, dass z.B. durch eine Konsistenzprüfung vor dem Aufruf der **launch**-Funktion die Aktivierung fehlerhafter DOS-Kommandos unterdrückt wird.

Da von der BAE-Grafikoberfläche keine Benutzereingaben an DOS-Kommandos übergeben werden können, ist dringend davon abzuraten, interaktive DOS-Kommandos bzw. Grafikapplikationen mit der **launch**-Funktion abzusetzen (andernfalls "hängt sich das System auf"). Unter UNIX lässt sich dieses Problem u.U. dadurch umgehen, dass interaktive Kommandos wie z.B. `more` oder `vi` in Hintergrundprozessen (&) aktiviert werden (was allerdings im Remote Login zu Problemen beim Terminalzugriff führen kann).

Siehe auch

Funktion **system**.

ldexp - Gleitkommamultiplikation mit 2^n (STD)

Synopsis

```
double ldexp(                   // Berechnungsergebnis
      double;                   // Eingabewert
      & int;                    // Exponent
      );
```

Beschreibung

Der Rückgabewert der Funktion **ldexp** entspricht dem Produkt von Eingabewert und der durch den Exponenten spezifizierten Potenz von zwei.

localtime - Systemdatum und Systemzeit abfragen (STD)

Synopsis

```
double localtime(               // CPU-Zeit (Sekunden)
      & int;                    // Sekunden seit Minutenanfang (0..59)
      & int;                    // Minuten seit Stundenanfang (0..59)
      & int;                    // Stunden seit Mitternacht (0..23)
      & int;                    // Tag (1..31)
      & int;                    // Monat (0..11)
      & int;                    // Jahre seit 1900
      & int;                    // Tage seit Sonntag (0..6)
      & int;                    // Tage seit Jahresanfang (0..365)
      );
```

Beschreibung

Die Funktion **localtime** gibt in den Parametern die aktuelle Systemzeit mit Datumsangabe zurück. Der Rückgabewert der Funktion entspricht der verbrauchten CPU-Zeit in Sekunden.

log - Logarithmus zur Basis e (STD)

Synopsis

```
double log(                     // Berechnungsergebnis
      double ]0.0,[;            // Eingabewert
      );
```

Beschreibung

Der Rückgabewert der Funktion **log** entspricht dem natürlichen Logarithmus (Basis e) des übergebenen Gleitkommawertes.

log10 - Logarithmus zur Basis 10 (STD)

Synopsis

```
double log10(                   // Berechnungsergebnis
      double ]0.0,[;            // Eingabewert
      );
```

Beschreibung

Der Rückgabewert der Funktion **log10** entspricht dem dekadischen Logarithmus (Basis 10) des übergebenen Gleitkommawertes.

mkdir - Dateiverzeichnis anlegen (STD)

Synopsis

```
int mkdir(                        // Status
      string;                     // Verzeichnispfadname
      );
```

Beschreibung

Die Funktion **mkdir** legt ein Dateiverzeichnis mit dem angegebenen Verzeichnispfadnamen an. Der Funktionsrückgabewert ist Null wenn das Verzeichnis erfolgreich erzeugt wurde oder (-1) bei fehlenden oder falschen Parametern oder bei Verzeichniszugriffsfehlern.

modf - Gleitkommazahl Vor- und Nachkommastellen (STD)

Synopsis

```
double modf(                      // Rückgabe Nachkommawert
      double;                     // Eingabewert
      & double;                   // Rückgabe Vorkommawert
      );
```

Beschreibung

Die Funktion **modf** teilt eine Gleitkommazahl in Vor- und Nachkommastellen auf. Der Rückgabewert der Funktion entspricht den Nachkommastellen. Die Vorkommastellen werden im entsprechenden Parameter zurückgegeben.

namestrcmp - Namensvergleich (STD)

Synopsis

```
int namestrcmp(                   // Vergleichsergebnis
      string;                     // Erster Name
      string;                     // Zweiter Name
      );
```

Beschreibung

Die Funktion **namestrcmp** vergleicht die beiden übergebenen Zeichenketten. Der Vergleich wird Zeicehn für Zeichen und ohne Unterscheidung von Groß- und Kleinschreibung vorgenommen. Der Rückgabewert ist Null bei Gleichheit, (-1) wenn die erste Zeichenkette kleiner als die zweite Zeichenkette ist und 1 wenn die erste Zeichenkette größer als die zweite Zeichenkette ist.

Siehe auch

Funktionen **numstrcmp**, **strcmp**.

numstrcmp - Numerischer Zeichenkettenvergleich (STD)

Synopsis

```
int numstrcmp(                    // Vergleichsergebnis
      string;                     // Erste Zeichenkette
      string;                     // Zweite Zeichenkette
      );
```

Beschreibung

Die Funktion **numstrcmp** vergleicht die beiden übergebenen Zeichenketten. Der Rückgabewert beträgt Null bei Gleichheit, (-1) wenn die erste Zeichenkette kleiner als die zweite Zeichenkette ist und ansonsten eins. Die Berechnung erfolgt für nicht Ziffern Zeichen für Zeichen durch numerischen Vergleich des ASCII-Wertes der verglichenen Zeichen. Bei Ziffern werden Zifferngruppen zu Zahlen zusammengefasst und diese Zahlen miteinander verglichen. Dadurch wird bei Sortierungen eine Reihenfolge von z.B. R1, R2, .. R10, R11 erreicht.

Siehe auch

Funktionen **namestrcmp**, **strcmp**.

perror - Fehlermeldung in Statuszeile ausgeben (STD)

Synopsis

```
void perror(
      string;                 // Zeichenkette
      );
```

Beschreibung

Die Funktion **perror** zeigt die übergebene Zeichenkette in der Statuszeile der BAE-Benutzeroberfläche an. Es erfolgt hierbei eine kurzzeitig invertierte Darstellung (einfaches Blinken), um den Anwender auf die Dringlichkeit der Meldung aufmerksam zu macheb. Bei Übergabe eines Leerstrings wird der Inhalt der Statuszeile gelöscht.

Siehe auch

Funktion **bae_prtdialog**.

pow - Potenzfunktion x^y (STD)

Synopsis

```
double pow(                   // Berechnungsergebnis
      double;                 // Basis
      double;                 // Exponent
      );
```

Beschreibung

Der Rückgabewert der Funktion **pow** entspricht dem Wert der mit dem Exponenten potenzierten Basis.

printf - Formatierte Ausgabe (STD)

Synopsis

```
void printf(
      string;                 // Formatzeichenkette
      []                      // Parameterliste
      );
```

Beschreibung

Die Funktion **printf** gibt die in der Parameterliste enthaltenen Daten formatiert im Arbeitsbereich des Bildschirms aus. Die Formatzeichenkette enthält Informationen, wie die Formatierung stattzufinden hat (siehe hierzu Beschreibung der Funktion **fprintf**).

Siehe auch

Funktionen **fprintf**, **sprintf**.

programid - Programmname abfragen (STD)

Synopsis

```
string programid(             // Programmname
      );
```

Beschreibung

Der Rückgabewert der Funktion **programid** entspricht dem Namen des gerade abgearbeiteten Programmes.

putchr - Zeichen ausgeben (STD)

Synopsis

```
int putchr(                     // Status
      char;                     // Zeichen
      );
```

Beschreibung

Die Funktion **putchr** gibt das übergebene Zeichen im Arbeitsbereich des Bildschirms aus. Der Rückgabewert der Funktion ist ungleich Null, wenn bei der Ausgabe ein Fehler aufgetreten ist.

putenv - Umgebungsvariable setzen (STD)

Synopsis

```
int putenv(                     // Status
      string;                   // Variablenname
      string;                   // Variablenwert
      );
```

Beschreibung

Die Funktion **putenv** weist der angegebenen Betriebssystemumgebungsvariablen einen Wert zu. Der Funktionsrückgabewert ist Null, wenn die Zuweisung erfolgreich war bzw. (-1) wenn die angegebene Variable nicht gefunden wurde.

Siehe auch

Funktion **getenv**.

puts - Zeichenkette mit Zeilenabschluss ausgeben (STD)

Synopsis

```
int puts(                       // Status
      string;                   // Zeichenkette
      );
```

Beschreibung

Die Funktion **puts** gibt die übergebene Zeichenkette im Arbeitsbereich des Bildschirms mit anschließendem Zeilenvorschub aus. Der Rückgabewert der Funktion ist ungleich Null, wenn bei der Ausgabe ein Fehler aufgetreten ist.

putstr - Zeichenkette ausgeben (STD)

Synopsis

```
int putstr(                     // Status
      string;                   // Zeichenkette
      );
```

Beschreibung

Die Funktion **putstr** gibt die übergebene Zeichenkette im Arbeitsbereich des Bildschirms aus. Der Rückgabewert der Funktion ist ungleich Null, wenn bei der Ausgabe ein Fehler aufgetreten ist.

quicksort - Indexliste sortieren (STD)

Synopsis

```
int quicksort(                    // Status
      & void;                     // Indexliste (Integer-Array)
      int;                        // Indexanzahl
      * int;                      // Elementvergleichsfunktion
      );
```

Beschreibung

Die Funktion **quicksort** sortiert die übergebene Indexliste nach dem Quicksortverfahren. Der Rückgabewert ergibt sich zu Null bei erfolgreicher Sortierung oder zu (-1) im Fehlerfall.

Elementvergleichsfunktion

```
int sortfuncname(
      int idx1,                   // Index 1
      int idx2,                   // Index 2
      )
{
      // Compare index 1 to index 2
      :
      return(compareresult);
}
```

Der Rückgabewert der Elementvergleichsfunktion sollte (-1) sein wenn der erste Indexwert kleiner ist als der zweite Indexwert, 1 wenn der erste Indexwert größer als der zweite ist, oder Null wenn beide Indexwerte gleich sind.

remove - Datei oder Verzeichnis löschen (STD)

Synopsis

```
int remove(                       // Status
      string;                     // Pfadname
      );
```

Beschreibung

Die Funktion **remove** löscht die Datei oder das Verzeichnis mit dem spezifizierten Pfadnamen. Der Rückgabewert der Funktion ist ungleich Null, wenn beim Löschen ein Fehler aufgetreten ist.

rename - Datei umbenennen (STD)

Synopsis

```
int rename(                       // Status
      string;                     // Alter Name
      string;                     // Neuer Name
      );
```

Beschreibung

Die Funktion **rename** ändert den Namen einer Datei. Der Rückgabewert der Funktion ist ungleich Null, wenn bei der Namensänderung ein Fehler aufgetreten ist.

rewind - Auf Dateianfang positionieren (STD)

Synopsis

```
void rewind(
      int;                        // Dateideskriptor
      );
```

Beschreibung

Die Funktion **rewind** positioniert den Dateizeiger der durch den Dateideskriptor beschriebenen Datei auf den Dateianfang.

rulecompile - Regeldefinition kompilieren (STD)

Synopsis

```
int rulecompile(                // Status
      string;                   // Zieldateiname
      string;                   // Regelname
      string;                   // Regelcode
      );
```

Beschreibung

Die Funktion **rulecompile** kompiliert den übergebenen Regelcode und speichert die kompilierte Regel unter dem angegebenen Regelnamen in der Zieldatei ab. Der Rückgabewert ist Null bei erfolgreicher Kompilierung oder ungleich Null wenn ein Fehler aufgetreten ist.

Siehe auch

Funktion **rulesource**.

rulesource - Regeldefinitionsquellcode abfragen (STD)

Synopsis

```
int rulesource(                 // Returns status
      string;                   // Regeldatenbankdateiname
      string;                   // Regelname
      & string;                   // Regelquellcode
      );
```

Beschreibung

Die Funktion **rulesource** ermittelt den Quellcode der durch Regeldatenbankdateiname und Regelname angegebenen Regeldefinition. Der Quellcode der Regeldefinition wird über einen entsprechenden Parameter als Zeichenkette an den Aufrufer zurückgegeben. Der Funktionsrückgabewert ist Null wenn die Abfrage erfolgreich war, (-1) bei fehlenden oder ungültigen Parametern, (-2) wenn die Regeldatenbank nicht gefunden/geöffnet werden konnte, (-3) wenn die Regeldefinition nicht gefunden wurde oder (-5) wenn die Regeldefinition nicht geladen werden konnte.

Siehe auch

Funktion **rulecompile**.

scanddbenames - Inhalt Datenbank abfragen (STD)

Synopsis

```
int scanddbenames(              // Scan Status
      string;                   // Dateiname
      int [100,[;               // Elementklasse (STD1)
      int [0,[;                 // Elementklasse (STD1) oder
                                // Null zum Auslesen des Cache
      & string;                 // Elementname Ein-/Ausgabe
      );
```

Beschreibung

Die Funktion **scanddbenames** sucht den dem übergebenen Feldnamen folgenden Elementeintrag der gegebenen Klasse in der Datenbankdatei und übergibt den Namen im Namensparameter. Bei Eingabe einer Leerzeichenkette wird das erste Element aus der Datenbank zurückgegeben. Der Rückgabewert ist 1 wenn ein Element gefunden wurde, 0 wenn kein weiteres Element in der Datenbankdatei vorhanden ist und (-1) wenn beim Dateizugriff Fehler aufgetreten sind oder die übergebenen Parameter ungültig sind.

scandirfnames - Inhalt Dateiverzeichnis abfragen (STD)

Synopsis

```
int scandirfnames(              // Scan Status
      string;                   // Verzeichnispfadname
      string;                   // Dateinamenserweiterung:
                                //    .EXT = Namensendung .EXT
                                //    .* = alle Dateien/Verzeichnisse
      & string;                 // Datei-/Verzeichnisname Ein-/Ausgabe
      );
```

Beschreibung

Die Funktion **scandirfnames** sucht den auf den übergebenen Namen folgenden Namenseintrag mit der gegebenen Endung im angegebenen Verzeichnis und übergibt den Namen im Namensparameter. Bei Eingabe einer Leerzeichenkette wird der erste Verzeichniseintrag zurückgegeben. Der Rückgabewert ist 1 wenn eine Datei bzw. ein Unterverzeichnis gefunden wurde, 0 wenn kein weiterer Eintrag im Verzeichnis vorhanden ist und (-1) wenn beim Verzeichniszugriff Fehler aufgetreten sind oder die übergebenen Parameter ungültig sind.

setprio - BAE Prozesspriorität setzen (STD)

Synopsis

```
void setprio(
      int;                      // Prozessprioritätswert
                                // - Unix/Linux:
                                //   nice-Prioritätswert
                                // - Windows:
                                //   <0 = HIGH_PRIORITY_CLASS
                                //    0 = NORMAL_PRIORITY_CLASS
                                //   >0 = IDLE_PRIORITY_CLASS
                                // - andernfalls ignoriert
      );
```

Beschreibung

Die Funktion **setprio** setzt die Priorität des aktuellen BAE-Prozesses entsprechend dem angegebenen Prozessprioritätswert.

sin - Sinus berechnen (STD)

Synopsis

```
double sin(                     // Berechnungsergebnis
      double;                   // Eingabewert (STD3)
      );
```

Beschreibung

Der Rückgabewert der Funktion **sin** entspricht dem Sinus des übergebenen Winkels. Der übergebene Gleitkommawert wird als Bogenmaßangabe interpretiert.

sinh - Hyperbolischen Sinus berechnen (STD)

Synopsis

```
double sinh(                    // Berechnungsergebnis
      double;                   // Eingabewert (STD3)
      );
```

Beschreibung

Der Rückgabewert der Funktion **sinh** entspricht dem hyperbolischen Sinus des übergebenen Winkels. Der übergebene Gleitkommawert wird als Bogenmaßangabe interpretiert.

sprintf - Formatierte Ausgabe auf String (STD)

Synopsis

```
int sprintf(                    // Anzahl dekodierte Zeichen
      & string;                 // Ausgabezeichenkette
      string;                   // Formatzeichenkette
      []                        // Parameterliste
      );
```

Beschreibung

Die Funktion **sprintf** gibt die in der Parameterliste enthaltenen Daten formatiert auf die Ausgabezeichenkette aus, d.h. mit dieser Funktion lässt sich eine Formatumwandlung im Speicher durchführen. Die Formatzeichenkette enthält Informationen, wie die Formatierung stattzufinden hat (siehe hierzu Beschreibung der Funktion **fprintf**). Der Rückgabewert der Funktion entspricht der Anzahl der umgewandelten Zeichen, d.h. die resultierende Länge der Ausgabezeichenkette.

Siehe auch

Funktionen **fprintf**, **printf**.

sqlcmd - SQL Kommando ausführen (STD)

Synopsis

```
int sqlcmd(                     // Status
      string;                   // Datenbankdateiname
      string;                   // SQL-Kommando
      * int;                    // Datenrückgabefunktion
      );
```

Beschreibung

Über die Funktion **sqlcmd** kann mit Hilfe einer Zugriffsprache der Datentransfer von und zu einer mit **sqlinit** initialisierten relationalen Datenbank gesteuert werden. Der erste Parameter der Funktion **sqlcmd** gibt den Namen der Datenbankdatei an, die für die Bearbeitung geöffnet werden soll. Im zweiten Parameter ist das auszuführende SQL-Kommando für den Datenzugriff zu spezifizieren. Die selektierten Datenfelder werden dem Aufrufer über die im dritten Parameter referenzierte Datenrückgabefunktion übergeben. Soll keine Datenrückgabefunktion aktiviert werden, dann ist für den entsprechenden Funktionsparameter das Schlüsselwort **`NULL`** einzutragen. Der Rückgabewert der Funktion **sqlcmd** ist ungleich Null bei fehlenden oder ungültigen Parametern oder wenn ein SQL-Datenbankfehler aufgetreten ist; die genaue Fehlerursache lässt sich im Falle eines SQL-Datenbankfehlers anschließend mit Hilfe der Funktion **sqlerr** bestimmen.

SQL-Kommandos

Die durch **sqlcmd** interpretierbare Datenbank-Zugriffssprache lehnt sich in ihrer Befehlssyntax an die Structured Query Language (SQL) für relationale Datenbanken an. Es werden die folgenden Grundbefehle angeboten:

`create table`
`drop table`
`insert into table values`
`quickinsert into table values`
`index table`
`select from table`
`delete from table`
`help`

Zur Repräsentation von Daten stehen die folgenden Datentypen zur Verfügung:

Schlüsselwort	Datentyp
`integer`	Ganze Zahlen im Bereich [-2147483648, 2147483647]
`float`	Fließpunktzahlen im Bereich [-10^308, 10^308] mit einer Genauigkeit von ca. 15 führenden Stellen
`string`	Zeichenketten (in einfachen Anführungszeichen)
`boolean`	Logical value (**`FALSE`** oder **`TRUE`**)
`date`	Datum; Eingabe **`dd/mm/yyyy`**, Ausgabe **`yyyymmdd`**

Kommando ***`create table`***

Mit dem **`create`**-Kommando wird eine Tabellenstruktur in der Datenbank hinterlegt. Dazu ist neben dem Tabellennamen eine Liste der Feldnamen mit den entsprechenden Datentypen der Feldelemente aufzuführen. Die in der Auflistung verwendete Reihenfolge wird auch bei der Datenausgabe verwendet, sofern keine explizite Angabe von Ausgabefeldern erfolgt. Die Syntax des **`create`**-Kommandos lautet:

```
create table tablename ( name1 type1, ..., namen typen ) ;
```

Für jedes Datenfeld einer Tabelle wird automatisch ein Index erstellt. Die Indizes werden bei der Bearbeitung von Datenbankabfragen - soweit dies sinnvoll ist - automatisch herangezogen. Der Benutzer braucht sich daher über deren optimale Verwendung keine Gedanken zu machen. Bei Stringvariablen werden nur die ersten 39 Zeichen für den Index berücksichtigt.

Kommando ***`drop table`***

Mit dem **`drop`**-Kommando wird eine Tabellenstruktur aus der Datenbankdatei entfernt. Dabei werden die Struktur der Tabelle sowie alle in der Tabelle gespeicherten Einträge gelöscht. Die Syntax des **`drop`**-Kommandos lautet:

```
drop table tablename ;
```

Kommando ***`insert into`***

Mit dem **`insert`**-Kommando wird ein Datensatz in eine Tabelle eingetragen. Die angegebenen Werte müssen in Anzahl und Typabfolge mit der Tabellendefinition übereinstimmen. Die Syntax des **`insert`**-Kommandos lautet:

```
insert into tablename values ( val1, ..., valn ) ;
```

Kommando ***`quickinsert insert into`***

Das Kommando **`quickinsert`** entspricht dem Kommando **`insert`** mit dem Unterschied, dass **`quickinsert`** im Gegensatz zu **`insert`** nach der Eintragung der Daten keinen automatischen Update der Feldindizes durchführt. Zur Aktualisierung der Feldindizes ist nach der (wiederholten) Anwendung des Kommandos **`quickinsert`** das Kommando **`index table`** auszuführen. **`quickinsert`** und **`index table`** eignen sich insbesondere zum schnellen Eintragen zahlreicher Datensätze in einer Tabelle, da die der einmalige Indizierungsprozess mit **`index table`** sehr viel schneller ist als die wiederholte Indizierung mit **`insert`**. Zu beachten ist dabei allerdings, dass mit **`quickinsert`** eingefügte Datensätze solange nicht in Abfrageergebnissen aufscheinen bis ein **`index table`**-Aufruf durchgeführt wurde.

Kommando ***`index table`***

Das Kommando **`index table`** dient dazu, zuvor mit **`quickinsert`** (siehe oben) in eine Tabelle eingespielten Datensätze zu indizieren. Die Syntax des **`index`**-Kommandos lautet:

```
index table tablename ;
```

Kommando `select from`

Mit dem **select**-Kommando können Datenbankinhalte abgefragt werden. Dabei kann über einen optional angebbaren **where**-Ausdruck die Ausgabemenge auf Datensätze beschränkt werden, für die die in diesem Ausdruck angegebenen Bedingungen erfüllt sind. Ist kein **where**-Ausdruck angegeben, dann erfolgt die Ausgabe sämtlicher Datensätze. Die Syntax des **select**-Kommandos lautet:

```
select [ field1, ..., fieldn ] from table1, ..., tablen
       [ where ... ] ;
```

Die Spezifikation der Ausgabefelder ist optional. Sind keine Ausgabefelder spezifiziert, dann werden sämtliche Ausgabefelder entsprechend der Reihenfolge ihrer Definition ausgegeben. Ist ein Ausgabefeldname in mehreren der angegebenen Tabellen definiert, so muss diese Mehrdeutigkeit durch eine Angabe des Feldnamens in der Form **table.field** aufgelöst werden. Dies gilt in gleicher Weise für die Referenzierung von Feldern innerhalb des **where**-Ausdrucks. Der **where**-Ausdruck setzt sich aus Vergleichen zusammen, die mit den logischen Operatoren **AND** (logisch Und), **OR** (logisch oder) und **NOT** (logisch nicht) verknüpft werden können. Die Vergleichsoperatoren **=** (gleich), **<>** (ungleich), **>** (größer), **>=** (größer gleich), **<** (kleiner) und **<=** (kleiner gleich) stehen für alle Datentypen zur Verfügung. Für den Stringdatentyp stehen darüber hinaus noch die Operatoren **PREVTO** (selektieren vorherigen Eintrag), **NEXTTO** (selektieren nächsten Eintrag) und **LIKE** (Stringmustervergleich) zur Verfügung; auf der rechten Seite dieser Operatoren kann ein Testpatternstring angegeben werden, der die Wildcardzeichen **%** (für beliebige Zeichenkette) und **?** (für beliebiges Einzelzeichen) enthalten darf. Bei Vergleichen müssen die Terme links und rechts der Vergleichsoperatoren typkompatibel, d.h. vom gleichen Datentyp sein; eine Ausnahme hiervon bildet lediglich die Datentypkombination **integer**-**float**. Terme können aus Operatoren und dadurch verknüpften Datenfeldreferenzen und Konstanten aufgebaut werden. Hierfür stehen die Operatoren **+** (Addition), **-** (Subtraktion), ***** (Multiplikation), **/** (Division), **%** (Rest nach Division), **UPPER** (Umwandlung in Großschreibung) und **LOWER** (Umwandlung in Kleinschreibung) zur Verfügung. Die Zulässigkeit der verwendbaren Operatoren hängt gemäß folgender Auflistung von den Datentypen der referenzierten Feldelemente bzw. Konstanten ab:

Operator	Datentyp
+	integer, float, string
-	integer, float
*	integer, float
/	integer, float
%	integer, float
UPPER()	string
LOWER()	string

Auf die Datentypen **date** und **boolean** kann keiner der definierten Operatoren angewendet werden.

Kommando `delete from`

Mit dem **delete**-Kommando werden Datensätze aus einer Tabellenstruktur entfernt. Die Syntax des **delete**-Kommandos lautet:

```
delete from tablename [ where ... ] ;
```

Die optional angebbare **where**-Bedingung entspricht in ihrer Syntax der **where**-Bedingung des **select**-Kommandos. Das **delete**-Kommando entfernt diejenigen Datensätze aus der angegebenen Tabelle, die der angegebenen **where**-Bedingung entsprechen; ist keine **where**-Bedingung angegeben, dann werden alle Datensätze aus der Tabelle entfernt.

Kommando cache

Das **cache**-Kommando dient dazu, die SQL-Datenbankdatei über mehrere Kommandos hinweg zum Lesen oder Schreiben geöffnet zu halten. Das Öffnen und Schliessen von SQL-Datenbankdateien ist insbesondere beim Zugriff auf Netzlaufwerke der zeitintensivste Teil bei Abfragen. Anwendungen die eine Vielzahl von Einzelkommandozugriffen auf eine SQL-Datenbank durchführen, können bei Verwendung dieser neuen Kommandos auf Netzlaufwerken um mehrere Größenordnungen schneller ablaufen. Die Syntax des **cache**-Kommandos unterstützt die folgenden Befehle:

```
cache read on ;
cache write on ;
cache off ;
```

Kommando help

Das **help**-Kommando dient dazu, Informationen über die in einer Datenbank definierten Tabellenstrukturen einzuholen. Die Syntax des **help**-Kommandos lautet:

```
help [ tablename ] ;
```

Die durch das **help**-Kommando ermittelten Daten werden über die Datenrückgabefunktion an den Aufrufer zurückgegeben. Die Angabe des Tabellennamens ist optional. Wird kein Tabellenname spezifiziert, dann liefert das **help**-Kommando die Namen der in der Datenbank definierten Tabellen im entsprechenden Parameter der Datenrückgabefunktion zurück, d.h. pro Tabellendefinition erfolgt ein Aufruf der Datenrückgabefunktion. Ist ein Tabellenname angegeben, dann liefert das **help**-Kommando die Namen der in dieser Tabelle definierten Datenfelder mit den entsprechenden Datentypangaben in den korrespondierenden Parametern der Datenrückgabefunktion zurück, d.h. pro Datenfeld der spezifizierten Tabelle erfolgt ein Aufruf der Datenrückgabefunktion.

Datenrückgabefunktion

```
int datafunc(
      string dstr,            // Zeichenkette oder Datum
      int dint,               // Ganzzahliger oder Logischer Wert
      double ddbl,            // Fließkommazahl
      int dval,               // Datengültigkeitskennzeichen:
                              //   0 = ungültige Daten
                              //   1 = gültige Daten
      int dtype,              // Datenfeld Typ:
                              //   2 = Ganzzahliger Wert
                              //   3 = Fließkommazahl
                              //   4 = Zeichenkette
                              //   5 = Datum (Format "yyyymmdd")
                              //   6 = Logischer Wert (0=FALSE,1=TRUE)
      string dtable,          // Tabellenname
      string dfield,          // Datenfeld Name
      int didx                // Datenausgabefeld Index
     )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Die Datenrückgabefunktion dient dazu, dem Aufrufer die selektierten Datenfelder zu übergeben. Die Funktion wird für jedes selektierte Datenfeld einzeln aufgerufen. Diese Funktion wird demnach bei z.B. 10 selektierten Datensätzen mit je 5 Datenfeldern 50 mal aufgerufen. Der Index des Datenausgabefeldes gibt an, das wievielte Datenfeld des aktuellen Datensatzes übergeben wird. Er nimmt Werte von 1 bis zur Anzahl der aktuell definierten Datenausgabefelder an. Der Rückgabewert der Datenrückgabefunktion sollte Null sein, wenn kein (semantischer) Fehler im Verarbeitungsprogramm der Funktion aufgetreten ist; im Fehlerfall sollte ein Wert ungleich Null zurückgegeben werden, um die Datenbankabfrage abzubrechen.

Warnung

Da die Funktion **sqlcmd** direkt auf den Datenbankdateiebene arbeitet, unterliegt sie - ebenso wie alle Dateizugriffsfunktionen des **AutoEngineers** - nicht dem Undo/Redo-Mechanismus.

Siehe auch

Funktionen **sqlerr**, **sqlinit**.

Beispiel

Erzeugen der Tabelle **`partdata`** mit den Datenfelddefinitionen **`symname`** (String), **`val`** (String) und **`partno`** (String) in der Datenbankdatei **`partdata.dat`**:

```
if (sqlinit("partdata.dat",1)!=0)
{
    perror("SQL Init error!");
    exit(0);
}
if (sqlcmd("partdata.dat",
 "create table partdata (symname string,val string,partno string);",
 NULL)!=0)
{
    perror("SQL Query error!");
    exit(0);
}
```

Eintragen von Daten in die Datenbank:

```
if (sqlcmd("partdata.dat",
 "insert into partdata values ('r','470','STK100470');",
 NULL)!=0)
{
    perror("SQL Data input error!");
    exit(0);
}
```

Abfragen von Daten:

```
if (sqlcmd("partdata.dat",
 "select partno from partdata where symname='r' AND val='470';",
 datafunc)!=0)
{
    perror("SQL Query error!");
    exit(0);
}
:
int datafunc(dstr,dint,ddbl,dval,dtype,dtable,dfield,didx)
string dstr;
int dint;
double ddbl;
int dval,dtype;
string dtable,dfield;
int didx;
{
        printf("Part Number : %s\n");
}
```

Löschen von Daten:

```
if (sqlcmd("partdata.dat",
 "delete from partdata where symname='r';",NULL)!=0)
{
    perror("SQL Delete error!");
    exit(0);
}
```

sqlerr - SQL Fehlerstatus abfragen (STD)

Synopsis

```
void sqlerr(
      & int;                  // Fehlercode
      & string;               // Fehlerstring
      );
```

Beschreibung

Die Funktion **sqlerr** dient der genauen Bestimmung der Fehlerursache nach einem erfolglosen Aufruf der Funktion **sqlcmd**.

Diagnose

Zur Bestimmung der Fehlerursache sind die durch **sqlerr** zurückgegebenen Parameterwerte heranzuziehen. Der zurückgegebene Fehlerstring identifiziert ggf. das fehlerverursachende Element. Die möglichen Werte, die der Fehlercode durch die Ausführung eines SQL-Kommandos annehmen kann, haben folgende Bedeutung:

Fehlercode	Bedeutung
0	SQL-Kommando fehlerfrei ausgeführt
1	SQL-Kommando Lesefehler (intern)
2	SQL-Kommando zu komplex (über 200 Terme)
3	Ungültiger numerischer Ausdruck
4	SQL-Kommandodatei nicht gefunden (intern)
5	SQL-Kommandoelement zu lang (über 200 Zeichen)
6	SQL-Kommando Syntaxfehler bei `<c>`
7	Allgemeiner SQL-Kommando-Parserfehler
8	Fehler beim Erzeugen der Datenbank
9	Dateizugriffsfehler
10	Zu viele offene Dateien
11	Datei `<d>` ist keine Datenbank
12	Datenbankstruktur beschädigt
13	Datenbankdateiaufbau fehlerhaft
14	Schlüsselbegriff `<k>` nicht gefunden
15	Schlüsselbegriff `<k>` existiert bereits
16	Datei `<d>` nicht gefunden
17	Ungültiger Datentyp für Tabellenelement `<t>.<f>`
18	Zu viele Tabellenelemente
19	Dateneintrag zu lang
20	Bedingte Löschung nur für eine Tabelle erlaubt
21	Term/Vergleich enthält unzulässige Typkombination
22	Ausgabefeld `<f>` undefiniert/in keiner Tabelle
23	Ausgabefeld `<f>` in mehreren Tabellen definiert
24	Ausgabetabelle `<t>` nicht in `from`-Tabellenliste
25	Tabelle `<t>` bereits definiert
26	Datenbankklassenanzahl überschreitet Datenbanklimit
27	Tabelle `<t>` nicht gefunden

28	Fehler von Datenrückgabefunktion
29	Kein **delete**-Record gefunden
30	Unbekanntes/neueres Datenbankformat
31	Abfragefeld nicht in Tabelle(n) enthalten
32	Abfragefeld in mehreren Tabellen enthalten
33	Dateilesezugriff verweigert
34	Dateischreibzugriff verweigert
35	Allgemeiner Datenbankfehler

Der Fehlerstring kann je nach Fehlerfall ein Kommandoelement **<c>**, eine Datei **<d>**, einen Schlüsselbegriff **<k>**, eine Tabelle **<t>** oder ein Datenfeld **<f>** bezeichnen.

Siehe auch

Funktionen **sqlcmd**, **sqlinit**.

sqlinit - SQL Datenbank initialisieren (STD)

Synopsis

```
int sqlinit(                    // Status
      string;                   // Datenbankdateiname
      int;                      // Initialisierungsmodus
      );
```

Beschreibung

Die Funktion **sqlinit** dient der Initialisierung eines mit den **sql***-Funktionen bearbeitbaren relationalen Datenbankssystems. Für den Initialisierungsmodus kann entweder 0 zur Benutzung einer existierenden Datenbankdatei, oder 1 zur Erzeugung einer neuen Datenbankdatei angegeben werden. Der Rückgabewert ist Null bei erfolgreicher Datenbankinitialisierung, oder 1 wenn die Datenbank bereits initialisiert ist; jeder andere Rückgabewert weist auf einen Fehler bei der Initialisierung bzw. Erzeugung der Datenbank hin. Das Format der mit den **sql***-Funktionen bearbeitbaren Dateien ist das DDB-Format des **Bartels AutoEngineer**. Das bedeutet, dass die mit den **sql***-Funktionen speicherbaren Daten in einer DDB-Datei zusammen mit AutoEngineer-Jobdaten oder in einer separaten DDB-Datei abgelegt werden können. Damit ist es z.B. möglich, projektrelevante PPS-Daten in der entsprechenden Designdatei abzulegen, oder spezielle Bauteilattribute automatisch mit entsprechenden Einträgen einer separaten Datenbank zu initialisieren. Für die relationale Datenbank sind im DDB-Format die Datenbankklassen 4096 bis 8191 reserviert. Dabei werden in der Datenbankklasse 4096 die Strukturen der einzelnen Tabellen hinterlegt. Die Datenbankklasse 4097 dient der Verwaltung der freien Datenbankklassen und enthält lediglich einen Eintrag mit dem Namen **info**, der bei der Initialisierung mit der Funktion **sqlinit** angelegt wird. Die Existenz dieses Eintrages ist ein eindeutiges Kriterium für die Abfrage, ob eine Datenbank für die Verwendung als relationale Datenbankdatei initialisiert ist. Die dynamisch verwalteten Datenbankklassen 4352 bis 8192 dienen der Speicherung der Tabelleneinträge und Indizes. Die Anzahl der für jede Tabelle benötigten Datenbankklassen ergibt sich aus der Anzahl der Tabellenfelder plus 1.

Siehe auch

Funktionen **sqlcmd**, **sqlerr**.

sqrt - Quadratwurzel berechnen (STD)

Synopsis

```
double sqrt(                    // Berechnungsergebnis
      double [0.0,[;            // Eingabewert
      );
```

Beschreibung

Der Rückgabewert der Funktion **sqrt** entspricht der Quadratwurzel des übergebenen Gleitkommawertes.

strcmp - ASCII-Zeichenkettenvergleich (STD)

Synopsis

```
int strcmp (                    // Vergleichsergebnis
      string;                   // Erste Zeichenkette
      string;                   // Zweite Zeichenkette
      );
```

Beschreibung

Die Funktion **strcmp** vergleicht die beiden übergebenen Zeichenketten. Der Rückgabewert beträgt Null bei Gleichheit, (-1) wenn die erste Zeichenkette kleiner als die zweite Zeichenkette ist und 1 ansonsten. Die Berechnung erfolgt Zeichen für Zeichen durch numerischen Vergleich der ASCII-Werte der beiden verglichenen Zeichen.

Siehe auch

Funktionen **namestrcmp**, **numstrcmp**.

strcspn - Zeichenkette Länge Startmuster berechnen (STD)

Synopsis

```
int strcspn(                    // Match Position
      string;                   // Eingabezeichenkette
      string;                   // Suchzeichenkette
      );
```

Beschreibung

Der Rückgabewert der Funktion **strcspn** entspricht der Anzahl der Zeichen am Beginn der Eingabezeichenkette die nicht in der übergebenen Suchzeichenkette vorhanden sind.

strdelchr - Zeichenkette Zeichenmenge entfernen (STD)

Synopsis

```
void strdelchr(
      & string;                 // Zeichenkette
      string;                   // Zu löschende Zeichen
      int;                      // Startposition
      int;                      // Endposition
      );
```

Beschreibung

Die Funktion **strdelchr** löscht in der übergebenen Zeichenkette von der Startposition bis zur Endposition alle Zeichen, die in der Zeichenkette der zu löschenden Zeichen aufgeführt sind.

strextract - Zeichenkette Sub-Zeichenkette extrahieren (STD)

Synopsis

```
string strextract(              // Extrahierte Zeichenkette
      string;                   // Zeichenkette
      int;                      // Startposition (0 - strlen-1)
      int;                      // Endposition (0 - strlen-1)
      );
```

Beschreibung

Die Funktion **strextract** gibt die bei der Startposition (Zählung beginnt bei 0) beginnende und bei der Endposition endende Sub-Zeichenkette der übergebenen Zeichenkette zurück. Start- und Endposition werden wenn nötig auf die Stringgrenzen justiert. Ist die Startposition größer als die Endposition, dann dreht sich auch die Extraktionsrichtung entsprechend um.

strextractfilepath - Verzeichnisname aus Dateipfadname extrahieren (STD)

Synopsis

```
string strextractfilepath(     // Rückgabe Verzeichnisname
```

```
      string;                 // Pfadname
      );
```

Beschreibung

Die Funktion **strextractfilepath** extrahiert den Verzeichnisnamen aus dem angegebenen Pfadnamen.

Siehe auch

Funktion **strgetpurefilename**.

strgetconffilename - Konfigurationsdateiname mit optionaler Umgebungsvariable bestimmen (STD)

Synopsis

```
string strgetconffilename(    // Rückgabe Konfigurationsdateipfadname
      string;                 // Bezeichnung Umgebungsvariable
      string;                 // Dateiname
      int;                    // Verzeichnisabfragemodus:
                              //    0 : Programmverzeichnis bevorzugen
                              //    1 : Datenverzeichnis für alle Benutzer
bevorzugen
                              //    2 : Datenverzeichnis für aktuellen Benutzer
bevorzugen
      );
```

Beschreibung

Die Funktion **strgetconffilename** gibt den über die angegebene Umgebungsvariable definierten Konfigurationsdateinamen zurück. Die Konfigurationsdateisuche wird dabei in verschiedenen Verzeichnissen entsprechend dem spezifierten Verzeichnisabfragemodus durchgeführt.

Siehe auch

Funktion **strgetvarfilename**.

strgetvarfilename - Dateiname aus Umgebungsvariable ableiten (STD)

Synopsis

```
string strgetvarfilename(     // Rückgabe Dateiname
      string;                 // Bezeichnung Umgebungsvariable
      );
```

Beschreibung

Die Funktion **strgetvarfilename** gibt den über die angegebene Umgebungsvariable definierten Datei- bzw. Dateipfadnamen zurück.

Siehe auch

Funktion **strgetconffilename**.

strgetpurefilename - Dateiname aus Dateipfadname extrahieren (STD)

Synopsis

```
string strgetpurefilename(    // Rückgabe Dateiname
      string;                 // Pfadname
      );
```

Beschreibung

Die Funktion **strgetpurefilename** extrahiert den Dateinamen aus dem angegebenen Pfadnamen.

Siehe auch

Funktion **strextractfilepath**.

strlen - Zeichenkette Länge ermitteln (STD)

Synopsis

```
int strlen(                      // Zeichenkettenlänge
      string;                    // Zeichenkette
      );
```

Beschreibung

Der Rückgabewert der Funktion **strlen** entspricht der Länge der übergebenen Zeichenkette (ohne abschließendes NUL-Zeichen).

strlistitemadd - String in Stringliste eintragen (STD)

Synopsis

```
void strlistitemadd(
      & string;                  // Durch Komma getrennte Stringliste
      string;                    // String
      );
```

Beschreibung

Die Funktion **strlistitemadd** trägt den spezifizierten String in die durch Komma getrennte Liste von Strings ein.

Siehe auch

Funktion **strlistitemchk**.

strlistitemchk - String in Stringliste suchen (STD)

Synopsis

```
int strlistitemchk(              // Suchergebnis
      string;                    // Durch Komma getrennte Stringliste
      string;                    // Suchstring
      );
```

Beschreibung

Die Funktion **strlistitemchk** sucht den spezifizierten Suchstring in der durch Komma getrennten Liste von Strings. Der Funktionsrückgabewert ist Null wenn der Suchstring nicht in der Stringliste enthalten ist, 1 wenn der Suchstring in der Stringliste enthalten ist oder 2 wenn der Suchstring mit der kompletten Stringliste identisch ist.

Siehe auch

Funktion **strlistitemadd**.

strlower - Zeichenkette in Kleinbuchstaben umwandeln (STD)

Synopsis

```
void strlower(
      & string;                  // Ein-/Ausgabezeichenkette
      );
```

Beschreibung

Die Funktion **strlower** wandelt die in der übergebenen Zeichenkette vorhandenen Buchstaben in Kleinbuchstaben um.

strmatch - Zeichenkette Musterabfrage (STD)

Synopsis

```
int strmatch(                  // Match Flag
      string;                  // Eingabezeichenkette
      string;                  // Testpatternzeichenkette
      );
```

Beschreibung

Die Funktion **strmatch** überprüft, ob die Eingabezeichenkette dem übergebenen Testpattern entspricht. Dabei sind in dem Testpattern die Sonderzeichen ***** für eine beliebige Zeichenfolge und **?** für ein beliebiges Zeichen in jeder Kombination mit normalen Zeichen erlaubt. Der Rückgabewert ist ungleich Null, wenn die Eingabezeichenkette dem Testpattern entspricht und Null wenn nicht.

strnset - Zeichenkette mit n Zeichen füllen (STD)

Synopsis

```
void strnset(
      & string;                // Zeichenkette
      char;                    // Füllzeichen
      int;                     // Füllanzahl
      );
```

Beschreibung

Die Funktion **strnset** ersetzt die ersten Füllanzahl Zeichen der übergebenen Zeichenkette mit dem Füllzeichen. Ist die Länge der Zeichenkette kleiner als die Füllanzahl, so werden nur die vorhandenen Zeichen, d.h. alle, ersetzt.

strreverse - Zeichenkette Zeichenreihenfolge umkehren (STD)

Synopsis

```
void strreverse(
      & string;                // Ein-/Ausgabezeichenkette
      );
```

Beschreibung

Die Funktion **strreverse** kehrt die Reihenfolge der Zeichen in der übergebenen Zeichenkette um.

strscannext - Zeichen in Zeichenkette vorwärts suchen (STD)

Synopsis

```
int strscannext(               // Match Position
      string;                  // Zeichenkette
      string;                  // Suchzeichenkette
      int;                     // Startposition (1 - strlen)
      int;                     // Stop-on-match Flag:
                               //    0 = Continue-on-match
                               //    1 = Stop-on-match
      );
```

Beschreibung

Die Funktion **strscannext** sucht in der übergebenen Zeichenkette ab der Startposition (Zählung beginnt bei 1) aufsteigend nach Zeichen, die in der Suchzeichenkette aufgeführt sind. Das Suchflag gibt an, ob die erste Position, die ein gesuchtes Zeichen enthält, zurückgegeben wird oder die erste Position, die kein gesuchtes Zeichen enthält. Der Rückgabewert gibt die gefundene Position an oder entspricht der Zeichenkettenlänge plus 1, wenn die Suche erfolglos war. Die Startposition wird wenn nötig auf die Stringgrenzen justiert.

strscanprior - Zeichen in Zeichenkette rückwärts suchen (STD)

Synopsis

```
int strscanprior(              // Match Position
      string;                  // Zeichenkette
      string;                  // Suchzeichenkette
      int;                     // Startposition
      int;                     // Stop-on-match Flag:
                               //    0 = Continue-on-match
                               //    1 = Stop-on-match
      );
```

Beschreibung

Die Funktion **strscanprior** sucht in der übergebenen Zeichenkette ab der Startposition absteigend nach Zeichen, die in der Suchzeichenkette aufgeführt sind. Das Suchflag gibt an, ob die erste Position, die ein gesuchtes Zeichen enthält, zurückgegeben wird oder die erste Position, die kein gesuchtes Zeichen enthält. Der Rückgabewert gibt die gefundene Position an oder Null wenn die Suche erfolglos war. Die Startposition wird wenn nötig auf die Stringgrenzen justiert.

strset - Zeichenkette mit Zeichen füllen (STD)

Synopsis

```
void strset(
      & string;                // Ein-/Ausgabezeichenkette
      char;                    // Füllzeichen
      );
```

Beschreibung

Die Funktion **strset** ersetzt alle Zeichen der übergebenen Zeichenkette mit dem Füllzeichen.

strspn - Zeichenkette Länge bis Endmuster ermitteln (STD)

Synopsis

```
int strspn(                    // Mismatch Position
      string;                  // Eingabezeichenkette
      string;                  // Suchzeichenkette
      );
```

Beschreibung

Der Rückgabewert der Funktion **strspn** entspricht der Anzahl der Zeichen am Beginn der Eingabezeichenkette die in der übergebenen Suchzeichenkette vorhanden sind.

strupper - Zeichenkette in Großbuchstaben umwandeln (STD)

Synopsis

```
void strupper(
      & string;                // Ein-/Ausgabezeichenkette
      );
```

Beschreibung

Die Funktion **strupper** wandelt die in der übergebenen Zeichenkette vorhandenen Buchstaben in Großbuchstaben um.

syngetintpar - BNF/Scanner Integerparameter abfragen (STD)

Synopsis

```
int syngetintpar(              // Status
      int [0,[;                // Parametertyp/-nummer:
                               //    0 = String-Kontrollzeichenauswertungsmodus
                               //    1 = Callback-Funktion für Kommentartexte
                               //         aktivieren
                               //    2 = Flag - beliebige Identifierzeichen
      & int;                   // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **syngetintpar** dient der Abfrage von mit **synsetintpar** im **User Language**-BNF-/Syntaxscanner gesetzten Integerparametern. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **synparsefile**, **synparseincfile**, **synparsestring**, **synsetintpar**.

synparsefile - BNF/Parser Datei einlesen (STD)

Synopsis

```
synparsefile(                  // Scan Status
      string;                  // Eingabedateiname
      [];                      // Optionale Callback-Funktion für Kommentare
      );
```

Beschreibung

Die Funktion **synparsefile** aktiviert einen Parser zur Abarbeitung der durch Dateinamen spezifizierten Eingabedatei. Dabei erfolgt die Abarbeitung entsprechend der im **User Language**-Programm festgelegten BNF-Beschreibung des Eingabedatenformats. Nach Bedarf werden automatisch die in der BNF definierten Aktionen bzw. Anwenderfunktionen ausgelöst. Innerhalb dieser Anwenderfunktionen kann mit **synscanline** die aktuelle Zeilennummer der Eingabedatei und mit **synscanstring** die aktuell eingelesene Zeichenkette ermittelt werden, wobei die aktuelle Zeichenkette nach Bedarf auch einer semantischen Prüfung unterzogen werden kann. Die Funktion **synparsefile** wird beendet, sobald das Ende der Eingabedatei erreicht ist oder ein Syntaxfehler bzw. ein durch eine Anwenderfunktion erkannter semantischer Fehler aufgetreten ist.

Callback-Funktion für Kommentare

Über den zweiten Funktionsparameter kann eine Callback-Funktion für Kommentartexte spezifiziert werden. Diese Callback-Funktion wird aktiviert, wenn zusätzlich der entsprechende Parameter über die Funktion **synsetintpar** gesetzt wurde. Die Callback-Funktion ist wie folgt zu definieren.

```
int commentfuncname(
      string commentstring,    // Kommentartext ohne Kommentarbegrenzer
      )
{
      // Verarbeitungsprogramm
      :
      return(stat);
}
```

Der Parserlauf wird abgebrochen, wenn die Callback-Funktion einen Wert ungleich Null zurückgibt. Andernfalls wird er Parserlauf forstgesetzt.

Diagnose

Die möglichen Rückgabewerte der Funktion **synparsefile** haben folgende Bedeutung:

Rückgabewert	Bedeutung
0	Parserlauf fehlerfrei beendet
1	Keine BNF-Definition verfügbar
2	Parser (**synparsefile**) ist bereits aktiv
3	Datei kann nicht geöffnet werden
4	Zu viele offene Dateien
5	Fataler Schreib-/Lesefehler
6	Scanelement zu lang
7	Syntaxfehler
8	Unerwartetes Dateiende
9	Stacküberlauf (BNF zu komplex)
10	Stackunterlauf (BNF fehlerhaft)
11	Fehler von Parser-Aktion/referenzierter Funktion

Siehe auch

Funktionen **syngetintpar**, **synparseincfile**, **synparsestring**, **synscaneoln**, **synscanigncase**, **synscanline**, **synscanstring**, **synsetintpar**, sowie Kapitel 2.6.4 dieses Handbuchs.

synparseincfile - BNF/Parser Includedatei einlesen (STD)

Synopsis

```
int synparseincfile(           // Status
      string;                  // Includedateiname
      );
```

Beschreibung

Mit der Parser-Funktion **synparseincfile** wird die Bearbeitung einer eingebundenen Datei (Includedatei) aktiviert, d.h. der Parser setzt bei Aufruf dieser Funktion den Einlesevorgang unmittelbar am Beginn der namentlich spezifizierten Includedatei fort. Sobald das Ende der Includedatei erreicht ist, wird das Terminalsymbol **EOFINC** als erkannt gemeldet, und der Einlesevorgang wird and der in der übergeordneten Datei unterbrochenen Stelle fortgesetzt. Der Rückgabewert von **synparseincfile** ist Null, wenn kein Fehler festgestellt wurde, (-1) wenn die Includedatei nicht geöffnet werden konnte, oder (-2) wenn der Parser, d.h. die Funktion **synparsefile** nicht aktiv ist.

Warnung

Bei Verwendung der Funktion **synparseincfile** ist eine entsprechende Einbindung des Terminalsymbols **EOFINC** in die BNF-Definition erforderlich, da der Parser sonst immer einen Syntaxfehler meldet, sobald das Ende der Includedatei erreicht ist.

Siehe auch
Funktionen **syngetintpar**, **synparsefile**, **synparsestring**, **synscaneoln**, **synscanigncase**, **synscanline**, **synscanstring**, **synsetintpar**, sowie Kapitel 2.6.4 dieses Handbuchs.

synparsestring - BNF/Parser Zeichenkette einlesen (STD)

Synopsis

```
synparsestring(                 // Scan Status
      string;                   // Eingabestring
      [];                       // Optionale Callback-Funktion für Kommentare
      );
```

Beschreibung

Die Funktion **synparsestring** aktiviert einen Parser zur Abarbeitung der übergebenen Zeichenkette. Dabei erfolgt die Abarbeitung entsprechend der im **User Language**-Programm festgelegten BNF-Beschreibung des Eingabedatenformats. Nach Bedarf werden automatisch die in der BNF definierten Aktionen bzw. Anwenderfunktionen ausgelöst. Innerhalb dieser Anwenderfunktionen kann mit **synscanline** die aktuelle Zeilennummer der Eingabedatei und mit **synscanstring** die aktuell eingelesene Zeichenkette ermittelt werden, wobei die aktuelle Zeichenkette nach Bedarf auch einer semantischen Prüfung unterzogen werden kann. Die Funktion **synparsestring** wird beendet, sobald das Ende der abzuarbeitenden Zeichenkette erreicht ist oder ein Syntaxfehler bzw. ein durch eine Anwenderfunktion erkannter semantischer Fehler aufgetreten ist.

Callback-Funktion für Kommentare

Über den zweiten Funktionsparameter kann eine Callback-Funktion für Kommentartexte spezifiziert werden. Diese Callback-Funktion wird aktiviert, wenn zusätzlich der entsprechende Parameter über die Funktion **synsetintpar** gesetzt wurde. Die Callback-Funktion ist wie folgt zu definieren.

```
int commentfuncname(
      string commentstring,   // Kommentartext ohne Kommentarbegrenzer
      )
{
      // Verarbeitungsprogramm
      :
      return(stat);
}
```

Der Parserlauf wird abgebrochen, wenn die Callback-Funktion einen Wert ungleich Null zurückgibt. Andernfalls wird er Parserlauf forstgesetzt.

Diagnose

Die möglichen Rückgabewerte der Funktion **synparsestring** haben folgende Bedeutung:

Rückgabewert	Bedeutung
0	Parserlauf fehlerfrei beendet
1	Keine BNF-Definition verfügbar
2	Parser (**synparsestring**) ist bereits aktiv
3	Datei kann nicht geöffnet werden
4	Zu viele offene Dateien
5	Fataler Schreib-/Lesefehler
6	Scanelement zu lang
7	Syntaxfehler
8	Unerwartetes Ende der Zeichenkette
9	Stacküberlauf (BNF zu komplex)
10	Stackunterlauf (BNF fehlerhaft)
11	Fehler von Parser-Aktion/referenzierter Funktion

Siehe auch

Funktionen **syngetintpar**, **synparsefile**, **synparseincfile**, **synscaneoln**, **synscanigncase**, **synscanline**, **synscanstring**, **synsetintpar**, sowie Kapitel 2.6.4 dieses Handbuchs.

synscaneoln - BNF/Scanner Zeilenendeerkennung setzen (STD)

Synopsis

```
int synscaneoln(                // Status
      int [0,1];                // Scanner Zeilenendeerkennung Modus:
                                //    0 = Deaktivieren EOLN-Erkennung
                                //    1 = Aktivieren EOLN-Erkennung
      );
```

Beschreibung

Mit der Funktion **synscaneoln** kann die Zeilenendeerkennung des durch **synparsefile** aktivierbaren BNF-Parsers wahlweise aktiviert oder deaktiviert werden. Per Default ist die Zeilenendeerkennung deaktiviert. Der Rückgabewert der Funktion ist ungleich Null bei Spezifikation ungültiger Parameter.

Warnung

Die Verwendung des Terminalsymbols `EOLN` in der BNF-Definition ist nur zulässig bzw. sinnvoll, wenn für die Dateibearbeitung auch die Zeilenendeerkennung aktiviert ist.

Siehe auch

Funktionen **synparsefile**, **synparseincfile**, **synparsestring**, **synscanigncase**, **synscanline**, **synscanstring**, sowie Kapitel 2.6.4 dieses Handbuchs.

synscanigncase - BNF/Scanner Schlüsselworterkennungsmodus setzen (STD)

Synopsis

```
int synscanigncase(             // Status
      int [0,1];                // Scanner Schlüsselworterkennungsmodus:
                                //    0 = Groß-/Kleinschreibung unterscheiden
                                //    1 = Groß-/Kleinschreibung ignorieren
      );
```

Beschreibung

Mit der Funktion **synscanigncase** kann die Unterscheidung zwischen Groß- und Kleinschreibung beim Einlesen von Schlüsselwörtern in dem durch **synparsefile** aktivierten BNF-Parser wahlweise deaktiviert oder aktiviert werden. Per Default unterscheidet der Parser zwischen Groß- und Kleinschreibung. Der Rückgabewert der Funktion ist ungleich Null bei Spezifikation ungültiger Parameter.

Siehe auch

Funktionen **synparsefile**, **synparseincfile**, **synparsestring**, **synscaneoln**, **synscanline**, **synscanstring**, sowie Kapitel 2.6.4 dieses Handbuchs.

synscanline - BNF/Scanner Eingabezeilennummer abfragen (STD)

Synopsis

```
int synscanline(                // Eingabezeilennummer
      );
```

Beschreibung

Der Rückgabewert der Funktion **synscanline** ist die Nummer der aktuell während des durch **synparsefile** aktivierten Parservorgangs eingelesene Eingabedateizeile.

Siehe auch

Funktionen **synparsefile**, **synparseincfile**, **synparsestring**, **synscaneoln**, **synscanigncase**, **synscanstring**, sowie Kapitel 2.6.4 dieses Handbuchs.

synscanstring - BNF/Scanner Eingabestring abfragen (STD)

Synopsis

```
string synscanstring(          // Scanner-Eingabestring
      );
```

Beschreibung

Der Rückgabewert der Funktion **synscanstring** ist die zuletzt während des durch **synparsefile** aktivierten Parservorgangs eingelesene Zeichenkette.

Siehe auch

Funktionen **synparsefile**, **synparseincfile**, **synparsestring**, **synscaneoln**, **synscanigncase**, **synscanline**, sowie Kapitel 2.6.4 dieses Handbuchs.

synsetintpar - BNF/Scanner Integerparameter setzen (STD)

Synopsis

```
int synsetintpar(              // Status
      int [0,[;                // Parametertyp/-nummer:
                               //    0 = String-Kontrollzeichenauswertungsmodus
                               //    1 = Callback-Funktion für Kommentartexte
                               //         aktivieren
                               //    2 = Flag - beliebige Identifierzeichen
      int;                     // Parameterwert
      );
```

Beschreibung

Die Funktion **synsetintpar** dient dazu, Systemparameter vom Typ im **`int`** im **User Language**-BNF-/Syntaxscanner zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **synsetintpar** gesetzten Systemparametern können mit der Funktion **syngetintpar** abgefragt werden.

Siehe auch

Funktionen **syngetintpar**, **synparsefile**, **synparseincfile**, **synparsestring**.

system - Betriebssystemkommando absetzen und Ausführung abwarten (STD)

Synopsis

```
int system(                      // Status
      string;                    // Kommando
      );
```

Beschreibung

Die Funktion **system** aktiviert das als Zeichenkette übergebene Kommando auf Betriebssystemebene. Das Kommando bzw. der Programmaufruf wird an den Kommandointerpreter des Betriebssystems, d.h. an die Betriebssystemshell übergeben, und die Ablaufkontrolle wird erst *nach* Ausführung des Kommandos wieder an BAE zurückgegeben. Der Rückgabewert der Funktion ergibt sich zu dem vom Betriebssystem, dem Kommandointerpreter bzw. dem aufgerufenen Programm zurückgegebenen Status. Ein Wert von Null deutet dabei in der Regel auf eine fehlerfreie Ausführung hin, während andere Werte üblicherweise Fehler oder Warnungen kennzeichnen.

Einschränkungen

Die Funktion **system** kann nicht in der **BAE Demo**-Software angewendet werden.

Unter MS-DOS benötigt der Phar Lap DOS-Extender ausreichend konventionellen Speicher zur Ausführung von (Child-)Prozessen. Die Größe des zur Verfügung stehenden konventionellen Speichers wird mit den Optionen **`-MINREAL`** und **`-MAXREAL`** des Phar Lap DOS Extenders festgelegt. Zur Ausführung von **User Language**-Programmen, die die **system**-Funktion benutzen, sind die **User Language**-Interpreterumgebungen mit Hilfe des mit der BAE-Software ausgelieferten CFIG386-Tools von Phar Lap wie folgt umzukonfigurieren:

```
> cfig386 <EXEFILE> -maxreal 0ffffh ↵
```

Für **`<EXEFILE>`** ist jeweils der Name des **User Language Interpreters** (**`scm.exe`**, **`ged.exe`**, **`neurrut.exe`**, **`cam.exe`**, **`gerview.exe`** bzw. **`ced.exe`**) einzusetzen.

Warnungen

Beachten Sie, dass die **system**-Funktion grundlegende Mehrprozess- bzw. Multitasking-Techniken voraussetzt, die auf PC-basierenden Systemen u.U. nicht ausreichend unterstützt werden, oder die in Rechnernetzwerken (abhängig vom auszuführenden Kommando) Probleme bereiten können.

Es wird dringend empfohlen, die Standardausgabe von DOS-Kommandos auf temporäre Dateien umzulenken und zur Anzeige eine **User Language**-Funktion zur Dateibetrachtung zu verwenden, da andernfalls die BAE-Grafikoberfläche von der Standardausgabe der mit **system** abgesetzten DOS-Kommandos überschrieben wird.

Die Fehlerausgabe von DOS-Kommandos erfolgt grundsätzlich auf den Bildschirm, wodurch die BAE-Grafikoberfläche überschrieben wird. Da unter DOS prinzipiell keine Möglichkeit zur Umlenkung der Fehlerausgabe besteht, ist dieses Problem nur dadurch zu umgehen, dass z.B. durch eine Konsistenzprüfung vor dem Aufruf der **system**-Funktion die Aktivierung fehlerhafter DOS-Kommandos unterdrückt wird.

Da von der BAE-Grafikoberfläche keine Benutzereingaben an DOS-Kommandos übergeben werden können, ist dringend davon abzuraten, interaktive DOS-Kommandos bzw. Grafikapplikationen mit der **system**-Funktion abzusetzen (andernfalls "hängt sich das System auf"). Unter UNIX lässt sich dieses Problem u.U. dadurch umgehen, dass interaktive Kommandos wie z.B. `more` oder `vi` in Hintergrundprozessen (&) aktiviert werden (was allerdings im Remote Login zu Problemen beim Terminalzugriff führen kann).

Siehe auch
Funktion **launch**.

tan - Tangens berechnen (STD)

Synopsis

```
double tan(                     // Berechnungsergebnis
      double;                   // Winkelwert (STD3)
      );
```

Beschreibung

Der Rückgabewert der Funktion **tan** entspricht dem Tangens des übergebenen Winkels. Der übergebene Gleitkommawert wird als Bogenmaßangabe interpretiert.

tanh - Hyperbolischen Tangens berechnen (STD)

Synopsis

```
double tanh(                    // Berechnungsergebnis
      double;                   // Winkelwert (STD3)
      );
```

Beschreibung

Der Rückgabewert der Funktion **tanh** entspricht dem hyperbolischen Tangens des übergebenen Winkels. Der übergebene Gleitkommawert wird als Bogenmaßangabe interpretiert.

tolower - Zeichen in Kleinbuchstaben umwandeln (STD)

Synopsis

```
char tolower(                   // Ausgabezeichen
      char;                     // Eingabezeichen
      );
```

Beschreibung

Der Rückgabewert der Funktion **tolower** ist der Großbuchstabe zu dem übergebenen Eingabezeichen. Bei Eingabezeichen, die keinen Buchstaben darstellen, wird der Eingabewert unverändert zurückgegeben.

toupper - Zeichen in Großbuchstaben umwandeln (STD)

Synopsis

```
char toupper(                   // Ausgabezeichen
      char;                     // Eingabezeichen
      );
```

Beschreibung

Der Rückgabewert der Funktion **toupper** ist der Kleinbuchstabe zu dem übergebenen Eingabezeichen. Bei Eingabezeichen, die keinen Buchstaben darstellen, wird der Eingabewert unverändert zurückgegeben.

uliptype - User Language Interpreterumgebung abfragen (STD)

Synopsis

```
int uliptype(                   // Interpretertyp:
                                //    0x0000 = Ungültig/unbekannt
                                //    0x0080 = SCM - Schaltplaneditor
                                //    0x0040 = GED - Layouteditor
                                //    0x0010 = AR - Autorouter
                                //    0x0008 = CAM - CAM-Prozessor
                                //    0x0004 = CED - Chipeditor
                                //    0x1000 = CV - CAM-View
      );
```

Beschreibung

Die Funktion **uliptype** gibt den Typ des aktuell aktiven **User Language Interpreter** zurück.

ulipversion - User Language Interpreter Version abfragen (STD)

Synopsis

```
int ulipversion(              // Interpreterversion
      );
```

Beschreibung

Die Funktion **ulipversion** gibt die interne Versionsnummer des aktuellen **User Language Interpreter** zurück.

ulproginfo - User Language-Programminfo abfragen (STD)

Synopsis

```
int ulproginfo(               // Status
      string;                 // Programmname
      & int;                  // Programmversion
      & int;                  // Programmaufruftyp
      );
```

Beschreibung

Die Funktion **ulproginfo** ermittelt die Version und die zulässigen Interpreterumgebungen des angegebenen **User Language**-Programms. Die Programmversion ist dabei die interne Versionsnummer des zur Kompilierung verwendeten **User Language Compilers**. Der Programmaufruftyp ist ein bit-maskierter Wert, über den sich die zur Programmausführung berechtigten **User Language**-Interpreterumgebungen abfragen lassen. Diese Angaben werden benötigt, um mit Hilfe der **ulip***-Funktionen zu entscheiden, ob ein spezielles **User Language**-Programm in der aktuell aktiven Interpreterumgebung ablauffähig ist. Der Rückgabewert der Funktion ist ungleich Null, wenn das angegebene Programm nicht gefunden wurde.

Siehe auch

Funktionen **uliptype**, **ulipversion**.

ulsystem - User Language-Programm aufrufen (STD)

Synopsis

```
int ulsystem(                    // Status
      string;                    // Programmname
      & int;                     // Programmzähler
      );
```

Beschreibung

Die Funktion **ulsystem** startet das angegebene **User Language**-Programm. Ein Rückgabewert ungleich Null gibt an, dass bei dem Laden oder Abarbeiten des Programmes ein Fehler aufgetreten ist. In dem Parameter Programmzähler wird im Fehlerfall die Adresse der zuletzt ausgeführten Anweisung des **User Language**=Programmes zurückgegeben (siehe Einträge in dem vom Compiler erzeugten Listingfile).

Diagnose

Die möglichen Rückgabewerte der Funktion **ulsystem** haben folgende Bedeutung:

Rückgabewert	Bedeutung
0	Programmlauf fehlerfrei beendet
1	DDB/Datenbank-Zugriffsfehler
2	Programm bereits geladen
3	Programm nicht gefunden
4	Inkompatible User Language Programm Version
5	Inkompatible Index-/Funktions-Referenzen
6	Stack Unterlauf
7	Stack Überlauf
8	Division durch Null
9	Fehler bei System-Funktions-Aufruf
10	System-Funktion nicht verfügbar
11	System-Funktion nicht implementiert
12	Anwender-Funktion nicht gefunden
13	Ungültiger Datentyp für referenzierte Funktion
14	Ungültige Parameterliste für referenzierte Funktion
15	Fehler beim Array-Zugriff
16	Ungültiger Array-Index
17	Allgemeiner Dateizugriffsfehler
18	Allgemeiner Dateilesefehler
19	Allgemeiner Dateischreibfehler

Siehe auch

Funktion **ulsystem_exit**.

ulsystem_exit - User Language-Programm nach Beendigung des aktuellen User Language-Programms aufrufen (STD)

Synopsis

```
void ulsystem_exit(
      string;                  // Programmname
      );
```

Beschreibung

Die Funktion **ulsystem** startet das angegebene **User Language**-Programm nach Beendigung des aktuellen **User Language**-Programms.

Siehe auch

Funktionen **exit**, **ulsystem**.

vardelete - Globale User Language-Variable löschen (STD)

Synopsis

```
int vardelete(                 // Status
      string;                  // Variablenname
      );
```

Beschreibung

Mit der Funktion **vardelete** kann eine zuvor mit **varset** definierte globale **User Language**-Variable gelöscht werden. Der Funktionsrückgabewert ist Null, wenn der Löschvorgang erfolgreich war oder (-1) wenn keine globale **User Language**-Variable mit dem spezifizierten Variablennamen definiert ist.

Siehe auch

Funktionen **varget**, **varset**.

varget - Globale User Language-Variable abfragen (STD)

Synopsis

```
int varget(                    // Status
      string;                  // Variablenname
      & void;                  // Variablenwert
      );
```

Beschreibung

Mit der Funktion **varget** kann der Wert einer zuvor mit **varset** definierten globalen **User Language**-Variablen ermittelt werden. Der Funktionsrückgabewert ist Null, wenn die Abfrage erfolgreich war, (-1) wenn keine globale **User Language**-Variable mit dem spezifizierten Variablennamen definiert ist oder (-2) wenn der Datentyp des Parameters für die Rückgabe des Variablenwertes nicht kompatibel zum aktuell definierten Datentyp der Variable ist.

Siehe auch

Funktionen **vardelete**, **varset**.

varset - Globale User Language-Variable setzen (STD)

Synopsis

```
int varset(                     // Status
      string;                   // Variablenname
      void;                     // Variablenwert
      );
```

Beschreibung

Die Funktion **varset** definiert eine globale **User Language**-Variable mit dem angegebenen Variablennamen und ordnet dieser Variable den spezifizierten Variablenwert zu. Der Variablenwert muss einen Basisdatentyp (**int**, **double**, **char** oder **string**) repräsentieren, d.h. es sind keine komplexen bzw. zusammengesetzten Datentypen wie etwa Arrays, Strukturen oder **index**-Typen zulässig. Der Funktionsrückgabewert ist Null bei erfolgreicher Variablendefinition, (-1) bei ungültigem Variablennamen oder (-2) bei unzulässigem Datentyp für den Variablenwert. Eine mit **varset** definierte Variable bleibt auch nach Beendigung des aktuell aktiven **User Language**-Programms definiert bis sie mit der Funktion **vardelete** wieder gelöscht werden oder bis das aktuell aktive BAE-Programm-Modul verlassen wird. Mit der Funktion **varget** können die Werte der mit **varset** definierten Variablen abgefragt werden. Da die mit **varset** definierten globalen Variablen auch nach Beendigung des definierenden **User Language**-Programms erhalten bleiben, ist über diese Variablen ein Datenaustausch zwischen zeitlich unabhängig voneinander ablaufenden **User Language**-Programmen möglich.

Siehe auch

Funktionen **vardelete**, **varget**.

C.3 SCM-Systemfunktionen

In diesem Abschnitt werden (in alphabetischer Reihenfolge) die in der **Bartels User Language** definierten SCM-Systemfunktionen beschrieben. Beachten Sie bitte die Konventionen zur Funktionsbeschreibung in Anhang C.1.

C.3.1 Schaltplan-Datenzugriffsfunktionen

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp CAP zugeordnet, d.h. diese Funktionen können im **Schaltplaneditor** aufgerufen werden:

cap_blockname - Schaltplan Blockname abfragen (CAP)

Synopsis

```
string cap_blockname(          // Schaltplan Blockname
      );
```

Beschreibung

Der Rückgabewert der Funktion **cap_blockname** ergibt sich aus dem Blocknamen, der für das aktuell geladene Stromlaufblatt definiert ist. Wenn kein Stromlaufblatt geladen ist oder wenn das aktuell geladene Element nicht als Blockschaltbild definiert ist, wird ein Leerstring zurückgegeben.

cap_blocktopflag - Schaltplan Blockhierarchieebene abfragen (CAP)

Synopsis

```
int cap_blocktopflag(          // Abfrageergebnis
      );
```

Beschreibung

Die Funktion **cap_blocktopflag** dient der Abfrage des Hierarchiemodus des aktuell geladenen Stromlaufblatts. Der Rückgabewert der Funktion **cap_blocktopflag** ergibt sich zu Null, wenn das aktuell geladene Schaltplanblatt als Blockschaltbild für einen hierarchischen Schatungsentwurf definiert ist, 1 wenn das Schaltplanblatt ein Planelement der obersten Stromlaufhierarchieebene darstellt, oder 2 wenn das Schaltplanblatt als einmalig referenzierbares Blockschaltbild definiert ist.

cap_figboxtest - SCM-Elementüberschneidung Rechteck prüfen (CAP)

Synopsis

```
int cap_figboxtest(            // Status
      index C_FIGURE;          // Element
      double;                  // Rechteck linke Grenze (STD2)
      double;                  // Rechteck untere Grenze (STD2)
      double;                  // Rechteck rechte Grenze (STD2)
      double;                  // Rechteck obere Grenze (STD2)
      );
```

Beschreibung

Die Funktion **cap_figboxtest** prüft, ob das angegebene Element das angegebene Rechteck schneidet. Der Rückgabewert ist ungleich Null, wenn die Elementgrenzen das angegebene Rechteck schneiden.

cap_findblockname - SCM-Blockschaltbild mit angegebenem Blocknamen suchen (CAP)

Synopsis

```
string cap_findblockname(     // Rückgabe SCM-Planelementname
      string;                 // DDB-Dateiname
      string;                 // Blockname
      );
```

Beschreibung

Die Funktion **cap_findblockname** sucht in der angegebenen DDB-Datei nach einem hierarchischen Schaltplanblatt mit dem spezifizierten Blocknamen und gibt den gefundenen SCM-Planelementnamen zurück. Ist der Zugriff auf die angegebene DDB-Datei nicht möglich oder existiert in der DDB-Datei kein hierarchisches Schaltplanblatt mit dem angegebenen Blocknamen, dann wird ein Leerstring zurückgegeben.

cap_findlayconpart - Bauteilindex aus Layoutnetzliste ermitteln (CAP)

Synopsis

```
int cap_findlayconpart(       // Status
      string;                 // Bauteilname
      & index CL_CPART;       // Rückgabe Bauteilindex
      );
```

Beschreibung

Die Funktion **cap_findlayconpart** sucht in der mit **cap_layconload** aktuell geladenen Layoutnetzliste nach dem spezifierten Bauteilnamen und gibt den zugehörigen Bauteilindex zurück. Der Funktionsrückgabewert ist Null wenn das Bauteil gefunden wurde oder ungleich Null bei fehlgeschlagener Suche.

Siehe auch

Funktionen **cap_findlayconpartpin**, **cap_findlaycontree**, **cap_getlaytreeidx**, **cap_layconload**.

cap_findlayconpartpin - Bauteilpinindex aus Layoutnetzliste ermitteln (CAP)

Synopsis

```
int cap_findlayconpartpin(    // Status
      string;                 // Pinname
      index CL_CPART;         // Netzlistenbauteil
      & index CL_CPIN;         // Rückgabe Netzlistenbauteilpin
      );
```

Beschreibung

Die Funktion **cap_findlayconpartpin** sucht in der mit **cap_layconload** aktuell geladenen Layoutnetzliste nach dem spezifierten Bauteilpinnamen und gibt den zugehörigen Bauteilpinindex zurück. Der Funktionsrückgabewert ist Null wenn der Bauteilpin gefunden wurde oder ungleich Null bei fehlgeschlagener Suche.

Siehe auch

Funktionen **cap_findlayconpart**, **cap_findlaycontree**, **cap_getlaytreeidx**, **cap_layconload**.

cap_findlaycontree - Netznamens-Netzindex aus Layoutnetzliste ermitteln (CAP)

Synopsis

```
int cap_findlaycontree(       // Status
      string;                 // Netzname
      & index CL_CNET;        // Rückgabe Netzindex
      );
```

Beschreibung

Die Funktion **cap_findlaycontree** sucht in der mit **cap_layconload** aktuell geladenen Layoutnetzliste nach dem spezifierten Netznamen und gibt den zugehörigen Netzindex zurück. Der Funktionsrückgabewert ist Null wenn das Netz gefunden wurde oder ungleich Null bei fehlgeschlagener Suche.

Siehe auch

Funktionen **cap_findlayconpartpin**, **cap_findlayconpart**, **cap_getlaytreeidx**, **cap_layconload**.

cap_getglobnetref - Globale Netznamensreferenz abfragen (CAP)

Synopsis

```
string cap_getglobnetref(      // Rückgabe Netznamensreferenz
                               //  (oder Leerstring wenn nicht referenziert)
      string;                  // Netz-/Treename
      );
```

Beschreibung

Die Funktion **cap_getglobnetref** ermittelt die globalen Netznamensreferenzbezeichnung für das angebene Netz.

cap_getlaytreeidx - Netznummer-Netzindex aus Layoutnetzliste ermitteln (CAP)

Synopsis

```
int cap_getlaytreeidx(         // Status
      int;                     // Netznummer
      & index CL_CNET;         // Rückgabe Netzindex
      );
```

Beschreibung

Die Funktion **cap_getlaytreeidx** sucht in der mit **cap_layconload** aktuell geladenen Layoutnetzliste nach der spezifierten Netznummer und gibt den zugehörigen Netzindex zurück. Der Funktionsrückgabewert ist Null wenn das Netz gefunden wurde oder ungleich Null bei fehlgeschlagener Suche.

Siehe auch

Funktionen **cap_findlayconpartpin**, **cap_findlayconpart**, **cap_findlaycontree**, **cap_layconload**.

cap_getpartattrib - SCM-Bauteilattributwert abfragen (CAP)

Synopsis

```
int cap_getpartattrib(         // Status
      string;                  // Bauteilname
      string;                  // Attributname
      & string;                // Attributwert Rückgabe
      );
```

Beschreibung

Die Funktion **cap_getpartattrib** ermittelt einen Attributwert eines namentlich spezifizierten Bauteils des aktuell geladenen Schaltplanblatts. Der Funktionsrückgabewert ist Null bei erfolgreicher Ermittlung des Attributwerts, (-1) wenn kein Stromlaufblatt geladen ist, (-2) bei fehlenden bzw. ungültigen Parametern, (-3) wenn das angegebene Bauteil auf dem aktuellen Schaltplan nicht gefunden wurde oder (-4) wenn das spezifizierte Attribut für das angegebene Bauteil nicht definiert bzw. nicht gesetzt ist.

cap_getrulecnt - SCM-Element Regelanzahl abfragen (CAP)

Synopsis

```
int cap_getrulecnt(             // Regelanzahl oder (-1) bei Fehler
      int;                      // Object class code
      int;                      // Object ident code (int oder Indextyp)
      );
```

Beschreibung

Mit der Funktion **cap_getrulecnt** kann die Anzahl der an ein spezifisches Objekt zugewiesenen Regeln ermittelt werden. Die Abfrage kann für das aktuell geladene Element (Objektklasse 0 mit **int**-Wert 0 für die Objektidentifikation), ein Element aus der Figurenliste des aktuell geladenen Elements (Objektklasse 1 mit gültigem Wert vom Typ index **C_FIGURE** für die Objektidentifikation) oder ein Poolelement (Objektklasse 2 mit gültigem Wert vom Typ index **C_POOL** für die Objektidentifikation) durchgeführt werden. Die von **cap_getrulecnt** ermittelte (nicht-negative) objektspezifische Regelanzahl wird im Rückgabewert der Funktion übergeben und bestimmt den Wertebereich für den Regelnamenslistenindex in nachfolgenden Aufrufen der Funktion **cap_getrulename** zur Ermittlung von Regelnamen für das entsprechende Objekt. Der Rückgabewert ergibt sich zu (-1), wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **cap_ruleerr** ermittelt werden.

Siehe auch

Funktionen **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_ruleplandet**, **cap_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

cap_getrulename - SCM-Element Regelname abfragen (CAP)

Synopsis

```
int cap_getrulename(            // Status
      int;                      // Object class code
      int;                      // Object ident code (int oder Indextyp)
      int [0,[;                 // Regelnamenslistenindex
      & string;                 // Regelname
      );
```

Beschreibung

Mit der Funktion **cap_getrulename** können die Namen der an ein spezifisches Objekt zugewiesenen Regeln ermittelt werden. Die Abfrage kann für das aktuell geladene Element (Objektklasse 0 mit **int**-Wert 0 für die Objektidentifikation), ein Element aus der Figurenliste des aktuell geladenen Elements (Objektklasse 1 mit gültigem Wert vom Typ index **C_FIGURE** für die Objektidentifikation) oder ein Poolelement (Objektklasse 2 mit gültigem Wert vom Typ index **C_POOL** für die Objektidentifikation) durchgeführt werden. Der Regelnamenslistenindex zur Auswahl der gewünschten Regel muss mindestens Null jedoch kleiner als die mit der Funktion **cap_getrulecnt** abfragbare Anzahl objektspezifischer Regeln sein. Der ermittelte Regelname wird über den letzten Funktionsparameter an den Aufrufer zurückgegeben. Der Rückgabewert der Funktion **cap_getrulename** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **cap_ruleerr** ermittelt werden.

Siehe auch

Funktionen **cap_getrulecnt**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_ruleplandet**, **cap_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

cap_getscbustapidx - Aktuell gescannten SCM-Busanschluß ermitteln (CAP)

Synopsis

```
index C_BUSTAP cap_getscbustapidx( // Busanschlußindex oder (-1) wenn kein
Busanschluß gescannt
      );
```

Beschreibung

Die Funktion **cap_getscbustappidx** gibt den aktuell gescannten Busanschlußindex zurück. Der Aufruf dieser Funktion ist nur innerhalb der Callbackfunktionen von **cap_scanall**, **cap_scanfelem** oder **cap_scanpool** sinnvoll. An anderer Stelle, oder wenn gerade kein Busanschluß gescannt wird, ergibt sich der Funktionsrückgabewert zu (-1).

Siehe auch

Funktionen **cap_scanall**, **cap_scanfelem**, **cap_scanpool**.

cap_getscclass - Aktuell gescannte SCM-Elementklasse ermitteln (CAP)

Synopsis

```
int cap_getscclass(           // Rückgabe SCM-Elementklasse:
                              //    0 = Schaltplan
                              //    1 = Symbol
                              //    2 = Marker
                              //    3 = Label
                              //    (-1) sonst
      );
```

Beschreibung

Die Funktion **cap_getscclass** gibt die aktuell gescannte SCM-Elementklasse zurück. Der Aufruf dieser Funktion ist nur innerhalb der Callbackfunktionen von **cap_scanall**, **cap_scanfelem** oder **cap_scanpool** sinnvoll. An anderer Stelle, oder wenn gerade kein SCM-Element gescannt wird, ergibt sich der Funktionsrückgabewert zu (-1).

Siehe auch

Funktionen **cap_scanall**, **cap_scanfelem**, **cap_scanpool**.

cap_getscrefpidx - Aktuell gescanntes SCM-Bibliothekselement ermitteln (CAP)

Synopsis

```
index C_POOL cap_getscrefpidx // Rückgabe Poolindex oder (-1) wenn kein Makro
      );
```

Beschreibung

Die Funktion **cap_getscrefpidx** gibt den Poolindex des aktuell gescannten Bibliothekselements zuück, d.h. damit kann beim Scannen von Polygonen, Texten, etc. die Zugehörigkeit des jeweils gescannten Objects zum entsprechenden Bibliothekselement (Symbol, Label, Marker) ermittelt werden. Der Aufruf dieser Funktion ist nur innerhalb der Callbackfunktionen von **cap_scanall**, **cap_scanfelem** oder **cap_scanpool** sinnvoll. An anderer Stelle, oder wenn gerade kein Bibliothekselement gescannt wird, ergibt sich der Funktionsrückgabewert zu (-1).

Siehe auch

Funktionen **cap_scanall**, **cap_scanfelem**, **cap_scanpool**.

cap_getscstkcnt - Schaltplan Scanfunktion Stacktiefe abfragen (CAP)

Synopsis

```
int cap_getscstkcnt(          // Scan-Stacktiefe
      );
```

Beschreibung

Die Funktion **cap_getscstkcnt** dient der Abfrage der aktuellen Stacktiefe der Schaltplanscanfunktionen. **cap_getscstkcnt** kann somit zur Abarbeitungskontrolle innerhalb der Callbackfunktionen von **cap_scanall**, **cap_scanfelem** oder **cap_scanpool** eingesetzt werden.

Siehe auch

Funktionen **cap_scanall**, **cap_scanfelem**, **cap_scanpool**.

cap_gettagdata - Schaltplan Tagsymbol Zieldaten abfragen (CAP)

Synopsis

```
int cap_gettagdata(           // Status
      index C_FIGURE;         // Tag Element
      int [0, [;              // Tag Index
      & int;                  // Tag Pintyp (CAP6)
      & string;               // Tag Pinname
      & string;               // Tag Referenzname 1
      & string;               // Tag Referenzname 2
      );
```

Beschreibung

Die Funktion **cap_gettagdata** dient der Ermittlung der Zieldaten (Pintyp, Pinname, Referenzbezeichnungen) des angegebenen Schaltplan-Tagsymbols. Der Funktionsrückgabewert ist Null bei erfolgreich Abfrage oder ungleich Null andernfalls.

Siehe auch

Funktion **scm_settagdata**.

cap_lastconseg - Zuletzt modifiziertes SCM-Verbindungssegment ermitteln (CAP)

Synopsis

```
int cap_lastconseg(           // Status
      & index C_CONSEG;       // Rückgabe Verbindungssegment
      );
```

Beschreibung

Die Funktion **cap_lastconseg** ermittelt das zuletzt erzeugte bzw. modifizierte SCM-Verbindungssegment und übergibt den entsprechenden Index aus der Verbindungssegmentliste im Rückgabeparameter. Der Rückgabewert der Funktion ist Null wenn ein derartiges Verdindungssegment existiert, oder ungleich Null andernfalls.

Siehe auch

Funktion **cap_lastfigelem**.

cap_lastfigelem - Zuletzt modifiziertes SCM-Element ermitteln (CAP)

Synopsis

```
int cap_lastfigelem(           // Status
      & index C_FIGURE;        // Rückgabe Element
      );
```

Beschreibung

Die Funktion **cap_lastfigelem** ermittelt das zuletzt erzeugte bzw. modifizierte SCM-Element und übergibt den entsprechenden Index aus der Figurenliste im Rückgabeparameter. Der Rückgabewert der Funktion ist Null wenn ein derartiges Element existiert, oder ungleich Null andernfalls.

Siehe auch

Funktion **cap_lastconseg**.

cap_layconload - Layoutnetzliste laden (CAP)

Synopsis

```
int cap_layconload(            // Status
      string;                  // DDB-Dateiname ("?" zur Namensabfrage)
      string;                  // Layout-Netzlistenname ("?" zur Namensabfrage)
      );
```

Beschreibung

Die Funktion **cap_layconload** lädt die Layoutnetzliste mit dem angegebenen Namen aus der spezifizierten DDB-Datei. Der Funktionsrückgabewert ist Null wenn die Netzliste erfolgreich geladen werden konnte, (-1) bei einem Dateizugriffsfehler oder (-2) bei fehlenden oder ungültigen Parameterangaben.

Siehe auch

Funktionen **cap_findlayconpartpin**, **cap_findlayconpart**, **cap_findlaycontree**, **cap_getlaytreeidx**.

cap_maccoords - Schaltplan Makrokoordinaten abfragen (CAP)

Synopsis

```
void cap_maccoords(
      & double;                // X-Position (STD2)
      & double;                // Y-Position (STD2)
      & double;                // Drehwinkel (STD3)
      & int;                   // Spiegelungsmodus (STD14)
      );
```

Beschreibung

Die Funktion **cap_maccoords** gibt in den Parametern die Platzierungsdaten für das aktuell bearbeitete Makro zurück. Ein Aufruf dieser Funktion ist nur innerhalb der Makroscanfunktion von **cap_scanall**, **cap_scanfelem** oder **cap_scanpool** sinnvoll. An anderer Stelle werden Null-Defaultwerte zurückgegeben.

Siehe auch

Funktionen **cap_scanall**, **cap_scanfelem**, **cap_scanpool**.

cap_macload - SCM-Symbol in den Arbeitsspeicher laden (CAP)

Synopsis

```
int cap_macload(                // Status
      & index C_POOL;           // Macro Poolelementindex
      string;                   // DDB-Dateiname
      string;                   // Elementname
      int [100,[;               // Element DDB-/Datenbankklasse (STD1)
      );
```

Beschreibung

Die Funktion **cap_macload** lädt das angegebene SCM-Symbol in den Arbeitsspeicher und übergibt den zugehörigen Poolindex im entsprechenden Parameter. Der Funktionsrückgabewert ist Null, wenn das Symbol erfolgreich geladen wurde, (-1) bei Dateizugriffsfehlern, (-2) bei fehlenden oder ungültigen Parameterangaben oder 1 wenn referenzierte Bibliothekselemente nicht verfügbar sind. **cap_macload** ist für die Anwendung in Funktionen zur Auswertung von Bibliotheksdateien konzipiert. Mit der Funktion **cap_macrelease** können SCM-Symbole wieder aus dem Arbeitsspeicher entfernt werden.

Siehe auch

Funktion **cap_macrelease**.

cap_macrelease - SCM-Symbol aus dem Arbeitsspeicher löschen (CAP)

Synopsis

```
void cap_macrelease(            // Status
      index C_POOL;             // Macro Poolelementindex
      );
```

Beschreibung

Die Funktion **cap_macrelease** löscht das über den Poolelementindex spezifizierte SCM-Symbol aus dem Arbeitsspeicher. **cap_macrelease** ist als Pendant zur Funktion **cap_macload** konzipiert.

Siehe auch

Funktion **cap_macload**.

cap_mactaglink - Schaltplan Makro-Tagverweisdaten abfragen (CAP)

Synopsis

```
int cap_mactaglink(             // Status
      & int;                    // Tagpintyp (CAP6)
      & double;                 // Startpunkt X-Position (STD2)
      & double;                 // Startpunkt Y-Position (STD2)
      & double;                 // Endpunkt X-Position (STD2)
      & double;                 // Endpunkt Y-Position (STD2)
      );
```

Beschreibung

Die Funktion **cap_mactaglink** gibt in den Parametern den Tagpintyp und die Start- und Endpunktkoordinaten des Tagverweises für das aktuell bearbeitete Makro zurück. Ein Aufruf dieser Funktion ist nur innerhalb der Makroscanfunktion von **cap_scanall**, **cap_scanfelem** oder **cap_scanpool** sinnvoll. Der Funktionsrückgabewert ist 1 wenn ein Tagverweis gefunden wurde, 0 wenn kein Tagverweis am bearbeiteten Makroelement existiert oder (-1) bei ungültigen oder fehlenden Parametern.

Siehe auch
Funktionen **cap_scanall**, **cap_scanfelem**, **cap_scanpool**.

cap_nrefsearch - Schaltplan Name auf Plan suchen (CAP)

Synopsis

```
int cap_nrefsearch(           // Status
      string;                 // Bauteilname oder Leerstring für neueste benannte
Referenz
      & index C_FIGURE;       // Rückgabe Element
      );
```

Beschreibung

Die Funktion **cap_nrefsearch** prüft, ob ein Bauteil bzw. eine benannte Referenz mit dem angegebenen Namen platziert ist und gibt gegebenenfalls das zugehörige Element zurück. Der Rückgabewert ist Null, wenn ein Element gefunden wurde oder ungleich Null, wenn kein Element mit dem angegebenen Namen gefunden wurde.

cap_partplan - Schaltplan Bauteilplanname abfragen (CAP)

Synopsis

```
string cap_partplan(          // Bauteilplanname
      string;                 // Job/DDB-Dateiname
      string;                 // Bauteilname
      );
```

Beschreibung

Der Rückgabewert der Funktion **cap_partplan** ergibt sich zu dem SCM-Plannamen, auf dem das durch den Bauteilnamen spezifizierte Bauteil in der angegebenen DDB-Datei platziert ist. Es wird ein Leerstring zurückgegeben, wenn das angegebene Bauteil nicht gefunden wurde.

cap_pointpoolidx - Schaltplan Verbindungspunktpoolelement ermitteln (CAP)

Synopsis

```
index C_POOL cap_pointpoolidx(// Poolelement
      );
```

Beschreibung

Die Funktion **cap_pointpoolidx** dient der Ermittlung des Poolelements, unter welchem die Daten des Markersymbols zur Darstellung der Verbindungspunkte auf dem aktuellen Stromlaufblatt gespeichert sind. Diese Funktion wird für das Plotten von Schaltplänen benötigt; die komplette Darstellungsform des Verbindungspunkt-Markers kann dabei nach Bedarf mit der Funktion **cap_scanpool** ermittelt werden.

cap_ruleconatt - Regelzuweisung an SCM-Verbindungssegment (CAP)

Synopsis

```
int cap_ruleconatt(           // Status
      index C_CONSEG;         // SCM-Verbindungssegment
      void;                   // Regelname oder Regelnamensliste
      );
```

Beschreibung

Die Funktion **cap_ruleconatt** erlaubt die Zuweisung von Regeln an das mit dem ersten Funktionsparameter spezifizierte SCM-Verbindungssegment des aktuell geladenes Elements. Der zweite Funktionsparameter erlaubt dabei sowohl die Spezifikation eines einzelnen Regelnamens (d.h. eines Wertes vom Typ `string`) als auch die Angabe einer ganzen Liste von Regelnamen (d.h. eines Arrays vom Typ `string`). Beachten Sie, dass vor der Zuweisung des angegebenen Regelsatzes zunächst alle bestehenden Regelzuweisungen an das spezifizierte Figurenlistenelement gelöscht werden. Der Rückgabewert der Funktion **cap_ruleconatt** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **cap_ruleerr** ermittelt werden.

Siehe auch

Funktionen **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_ruleplandet**, **cap_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

cap_rulecondet - Regelzuweisungen von SCM-Verbindungssegment lösen (CAP)

Synopsis

```
int cap_rulecondet(           // Status
      index C_CONSEG;         // SCM-Verbindungssegment
      );
```

Beschreibung

Die Funktion **cap_rulecondet** löscht *alle* aktuell bestehenden Regelzuweisungen an das über den Funktionsparameter spezifizierte SCM-Verbindungssegment des aktuell geladenen Elements. Der Rückgabewert der Funktion **cap_rulecondet** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **cap_ruleerr** ermittelt werden.

Siehe auch

Funktionen **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**. **cap_ruleplandet**, **cap_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

cap_ruleerr - SCM-Regelsystem Fehlerstatus abfragen (CAP)

Synopsis

```
void cap_ruleerr(
      & int;              // Fehlercode
      & string;           // Fehlerstring
      );
```

Beschreibung

Die Funktion **cap_ruleerr** dient der Ermittlung des Regelsystemstatus, d.h. die Funktion **cap_ruleerr** kann zur genauen Bestimmung der Fehlerursache im Falle eines fehlerhaften Aufrufs einer Regelsystemfunktion verwendet werden.

Diagnose

Zur Bestimmung der Fehlerursache sind die durch **cap_ruleerr** zurückgegebenen Parameterwerte heranzuziehen. Der zurückgegebene Fehlerstring dient ggf. der Identifizierung des fehlerverursachenden Elements. Die möglichen Werte, die der Fehlercode durch die Ausführung eines Regelsystemfunktion annehmen kann, haben folgende Bedeutung:

Fehlercode	Bedeutung
0	Regelsystem Operation/Funktion erfolgreich beendet
1	Regelsystem Hauptspeicher nicht ausreichend
2	Regelsystem Interner Fehler **<e>**
3	Regelsystem Funktionsparameter ungültig
128	Regelsystem Datenbankdatei kann nicht angelegt werden
129	Regelsystem Datenbankdatei Lese-/Schreibfehler
130	Regelsystem Datenbankdatei von falschem Typ
131	Regelsystem Datenbankdateistruktur beschädigt
132	Regelsystem Datenbankdatei nicht gefunden
133	Regelsystem Datenbankfehler allgemein (Interner Fehler)
134	Regelsystem Regel **<r>** nicht Regeldatenbank gefunden
135	Regelsystem Regel in falschem Format in Datenbank (Interner Fehler **<e>**)
136	Regelsystem Objekt nicht gefunden
137	Regelsystem Objekt mehrfach definiert (Interner Fehler)
138	Regelsystem Inkompatible Definition der Variable **<v>**
139	Regelsystem Regel **<r>** mit inkompatibler Compiler-Version übersetzt

Der Fehlerstring kann je nach Fehlerfall eine Regel **<r>**, eine Variable **<v>** oder einen (internen) Fehlerstatus **<e>** bezeichnen. Datenbankdateifehler beziehen sich auf Probleme beim Zugriff auf die Regeldatenbankdatei **`brules.vdb`** im BAE-Programmverzeichnis. Interne Fehler weisen üblicherweise auf Implementierungslücken im Regelsystem hin und sollten in jedem Fall an Bartels gemeldet werden.

Siehe auch

Funktionen **cap_getrulecnt**, **cap_getrulename**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_ruleplandet**, **cap_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

cap_rulefigatt - Regelzuweisung an SCM-Figurenelement (CAP)

Synopsis

```
int cap_rulefigatt(             // Status
      index C_FIGURE;           // Figurenlistenelement
      void;                     // Regelname oder Regelnamensliste
      );
```

Beschreibung

Die Funktion **cap_rulefigatt** erlaubt die Zuweisung von Regeln an das mit dem ersten Funktionsparameter spezifizierte Figurenlistenelement des aktuell geladenes Elements. Der zweite Funktionsparameter erlaubt dabei sowohl die Spezifikation eines einzelnen Regelnamens (d.h. eines Wertes vom Typ `string`) als auch die Angabe einer ganzen Liste von Regelnamen (d.h. eines Arrays vom Typ `string`). Beachten Sie, dass vor der Zuweisung des angegebenen Regelsatzes zunächst alle bestehenden Regelzuweisungen an das spezifizierte Figurenlistenelement gelöscht werden. Der Rückgabewert der Funktion **cap_rulefigatt** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **cap_ruleerr** ermittelt werden.

Siehe auch

Funktionen **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_ruleplandet**, **cap_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

cap_rulefigdet - Regelzuweisungen von SCM-Figurenelement lösen (CAP)

Synopsis

```
int cap_rulefigdet(             // Status
      index C_FIGURE;           // Figurenlistenelement
      );
```

Beschreibung

Die Funktion **cap_rulefigdet** löscht *alle* aktuell bestehenden Regelzuweisungen an das über den Funktionsparameter spezifizierte Figurenlistenelement des aktuell geladenen Elements. Der Rückgabewert der Funktion **cap_rulefigdet** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **cap_ruleerr** ermittelt werden.

Siehe auch

Funktionen **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_ruleplanatt**. **cap_ruleplandet**, **cap_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

cap_ruleplanatt - Regelzuweisung an aktuell geladenes SCM-Element (CAP)

Synopsis

```
int cap_ruleplanatt(          // Status
      void;                   // Regelname oder Regelnamensliste
      int [0,1];              // Flag - Regelzuweisung SCM-Plan global
      );
```

Beschreibung

Die Funktion **cap_ruleplanatt** erlaubt die Zuweisung von Regeln an das aktuell geladenes Elements. Der Funktionsparameter erlaubt dabei sowohl die Spezifikation eines einzelnen Regelnamens (d.h. eines Wertes vom Typ **string**) als auch die Angabe einer ganzen Liste von Regelnamen (d.h. eines Arrays vom Typ **string**). Beachten Sie, dass vor der Zuweisung des angegebenen Regelsatzes zunächst alle bestehenden Regelzuweisungen an das aktuelle Element gelöscht werden. Der zweite Parameter ermöglicht die Zuweisung von Regeln an *alle* Blätter des aktuell geladenen Schaltplans (d.h., dieser Parameter wird nur ausgewertet, wenn ein Stromlaufblatt geladen ist). Der Rückgabewert der Funktion **cap_ruleplanatt** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **cap_ruleerr** ermittelt werden.

Siehe auch

Funktionen **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplandet**, **cap_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

cap_ruleplandet - Regelzuweisungen von aktuell geladenem SCM-Element lösen (CAP)

Synopsis

```
int cap_ruleplandet(          // Status
      int [0,1];              // Flag - Regelzuweisung SCM-Plan global
      );
```

Beschreibung

Die Funktion **cap_ruleplandet** löscht *alle* aktuell bestehenden Regelzuweisungen an das aktuell geladene Element. Der Funktionsparameter ermöglicht die Lösung von Regelzuweisungen von *allen* Blätter des aktuell geladenen Schaltplans (d.h., dieser Parameter wird nur ausgewertet, wenn ein Stromlaufblatt geladen ist). Der Rückgabewert der Funktion **cap_ruleplandet** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **cap_ruleerr** ermittelt werden.

Siehe auch

Funktionen **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

cap_rulequery - SCM-Element Regelabfrage durchführen (CAP)

Synopsis

```
int cap_rulequery(              // Trefferanzahl oder (-1) bei Fehler
      int;                      // Object class code
      int;                      // Object ident code (int oder Indextyp)
      string;                   // Subjektname
      string;                   // Prädikatname
      string;                   // Abfragekommando
      & void;                   // Abfrageergebnis
      []                        // Optionale Abfrageparameter
      );
```

Beschreibung

Die Funktion **cap_rulequery** führt eine Regelabfrage für ein spezifisches Objekt durch. Die Abfrage kann für das aktuell geladene Element (Objektklasse 0 mit **int**-Wert 0 für die Objektidentifikation), ein Element aus der Figurenliste des aktuell geladenen Elements (Objektklasse 1 mit gültigem Wert vom Typ index **C_FIGURE** für die Objektidentifikation) oder ein Poolelement (Objektklasse 2 mit gültigem Wert vom Typ index **C_POOL** für die Objektidentifikation) durchgeführt werden. Zur Durchführung der Abfrage müssen sowohl ein Regelsubjekt als auch ein Regelprädikat namentlich angegeben werden. Zusätzlich ist ein Abfragekommando zu spezifizieren. Das Abfragekommando kann Platzhalter für Wertvorgaben und einen Abfrageoperator enthalten. Folgende Abfrageoperatoren stehen zur Verfügung:

?d	zur Abfrage von **int**-Werten
?f	zur Abfrage von **double**-Werten
?s	zur Abfrage von **string**-Werten

Dem Abfrageoperator kann wahlweise einer der folgenden Selektionsoperatoren vorangestellt werden:

+	zur Abfrage des Maximums aller gefundenen Werte
-	zur Abfrage des Minimums aller gefundenen Werte

Standardmäßig, d.h. bei Auslassung des Selektionsoperators wird der **+**-Operator verwendet. Der über die Abfrage gefundene Werteintrag wird im Funktionsparameter für das Abfrageergebnis zurückgegeben. Hierbei ist sicherzustellen, dass der Datentyp des Parameters für das Abfrageergebnis mit dem Abfragedatentyp übereinstimmt (**int** für **?d**, **double** für **?f**, **string** für **?s**). Neben dem Abfrageoperator können folgende Platzhalter für Wertvorgaben im Abfragekommando spezifiziert werden:

%d	zur Angabe von **int**-Werten
%f	zur Angabe von **double**-Werten
%s	zur Angabe von **string**-Werten

Für jeden im Abfragekommando spezifizierten Platzhalter für Wertvorgaben ist ein optionaler Abfrageparameter an die Funktion **cap_rulequery** zu übergeben. Die Reihenfolge dieser optionalen Parameter sowie deren Datentypen müssen mit den Spezifikationen im Abfragekommando übereinstimmen. Nach erfolgreicher Abarbeitung der Regelabfrage wird im Rückgabewert die (nicht-negative) Anzahl der gefundenen Einträge an den Aufrufer zurückgegeben. Der Rückgabewert ergibt sich zu (-1), wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **cap_ruleerr** ermittelt werden.

Beispiele

Sofern die Regel

```
rule somerule
{
    subject subj
    {
        pred := ("A", 2);
        pred := ("A", 4);
        pred := ("B", 1);
        pred := ("C", 3);
        pred := ("B", 6);
        pred := ("D", 5);
        pred := ("D", 6);
        pred := ("A", 3);
    }
}
```

definiert und dem aktuell geladenen Element zugewiesen ist, würde der **cap_rulequery**-Aufruf

```
hitcount = cap_rulequery(0,0,"subj","pred","%s ?d",intresult,"A") ;
```

die **`int`**-Variable **`hitcount`** auf 3 und die **`int`**-Variable **`intresult`** auf 4 setzen, während der Aufruf

```
hitcount = cap_rulequery(0,0,"subj","pred","-?s %d",strresult,6) ;
```

die Variable **`hitcount`** auf 2 und die **`string`**-Variable **`strresult`** auf **`B`** setzt.

Siehe auch

Funktionen **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_ruleplandet**; **Neuronales Regelsystem** und Rule System Compiler.

cap_scanall - Schaltplan Scan über alle Elemente (CAP)

Synopsis

```
int cap_scanall(                // Scan Status
      double;                   // X-Offset (STD2)
      double;                   // Y-Offset (STD2)
      double;                   // Drehwinkel (STD3)
      int [0,1];                // Element in Arbeitsbereich Flag (STD10)
      * int;                    // Makrofunktion
      * int;                    // Verbindungssegmentfunktion
      * int;                    // Polygonfunktion
      * int;                    // Textfunktion
      );
```

Beschreibung

Die Funktion **cap_scanall** scannt alle auf dem aktuell geladenen Element platzierten Elemente über alle Hierarchiestufen. Dabei werden für alle gefundenen Elemente die referenzierten Anwenderfunktionen aufgerufen. Soll ein Funktionstyp nicht aufgerufen werden, so ist für den entsprechenden Parameter `NULL` anzugeben. Der Rückgabewert der Funktion **cap_scanall** ist ungleich Null, wenn ungültige Parameter angegeben wurden, oder wenn eine der referenzierten Anwenderfunktionen einen Fehler an die Funktion **cap_scanall** zurückgemeldet hat.

Makrofunktion

```
int macrofuncname(
      index C_MACRO macro,      // Makro Index
      index C_POOL pool,        // Pool Element Index
      int macinws,              // Makro in Arbeitsbereich Flag (STD10)
      string refname,           // Makro Referenzname
      index C_LEVEL level,      // Makro Level
      index C_LEVEL buslevel    // Makro Buslevel
      )
{
      // Verarbeitungsprogramm
      :
      return(contscan);
}
```

Miteinander verbundene Elemente besitzen gleiche Levelnummern größer gleich Null. Der Rückgabewert dieser Funktion sollte gleich Null sein, wenn der Scan für dieses Makro nicht weitergeführt werden soll, (-1) wenn ein Fehler aufgetreten ist und 1 wenn weitergescannt werden soll.

Verbindungssegmentfunktion

```
int confuncname(
      index C_CONBASE cbase,  // Verbindungsliste Index
      int segidx,             // Segment Nummer
      int ctyp,               // Verbindungstyp:
                              //    0 = normale Verbindung
                              //    1 = Verbindungspunkt
      double lx,              // Untere X-Koordinate (STD2)
      double ly,              // Untere Y-Koordinate (STD2)
      double ux,              // Obere X-Koordinate (STD2)
      double uy,              // Obere Y-Koordinate (STD2)
      int busflag,            // Busverbindung Flag:
                              //    0 = normale Verbindung
                              //    1 = Busverbindung
      int cinws,              // Verbindung in Arbeitsbereich (STD10)
      index C_LEVEL level     // Verbindung Level
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Ein Verbindungstyp von 0 gibt an, dass es sich um eine normale Segmentverbindung handelt. Ein Typ von 1 spezifiziert einen Verbindungskreuzungspunkt, obere und untere Koordinaten sind dabei identisch. Miteinander verbundene Elemente besitzen gleiche Levelnummern größer gleich Null. Der Rückgabewert dieser Funktion sollte ungleich Null sein, wenn ein Fehler aufgetreten ist.

Polygonfunktion

```
int polyfuncname(
      index C_POLY poly,      // Polygondaten
      int polyinws,           // Polygon in Arbeitsbereich Flag (STD10)
      index C_LEVEL level,    // Polygon Level
      int macclass,           // Polygon Makroklasse (STD1)
      int bustapidx           // Polygon Busanschlussindex
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Die Makroklasse gibt die Klasse des Makros an, auf dem die Fläche platziert ist. Miteinander verbundene Elemente besitzen gleiche Levelnummern größer gleich Null. Ein Busanschlussindex größer gleich Null gibt an, dass es sich um eine auf einem Bustap platzierte Fläche handelt. Der Rückgabewert dieser Funktion sollte ungleich Null sein, wenn ein Fehler aufgetreten ist.

Textfunktion

```
int textfuncname(
      index C_TEXT text,        // Textdaten
      double x,                 // X-Koordinate (STD2)
      double y,                 // Y-Koordinate (STD2)
      double angle,             // Drehwinkel (STD3)
      int mirr,                 // Spiegelung (STD14)
      double size,              // Text Größe (STD2)
      string textstr,           // Textzeichenkette
      int textinws,             // Text in Arbeitsbereich Flag (STD10)
      int macclass,             // Text Makroklasse
      int variant               // Text ist Variantenattribut Flag
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Die Makroklasse gibt die Klasse des Makros an, auf dem der Text platziert ist. Der Rückgabewert dieser Funktion sollte ungleich Null sein, wenn ein Fehler aufgetreten ist.

Siehe auch

Funktionen **cap_maccoords**, **cap_scanfelem**, **cap_scanpool**.

cap_scanfelem - Schaltplan Scan über Figurenelement (CAP)

Synopsis

```
int cap_scanfelem(              // Scan Status
      index C_FIGURE;           // Figurenelement
      double;                   // X-Offset (STD2)
      double;                   // Y-Offset (STD2)
      double;                   // Drehwinkel (STD3)
      int [0,1];                // Element in Arbeitsbereich Flag (STD10)
      * int;                    // Makrofunktion
      * int;                    // Verbindungssegmentfunktion
      * int;                    // Polygonfunktion
      * int;                    // Textfunktion
      );
```

Beschreibung

Die Funktion **cap_scanfelem** scannt das angegebene Figurenelement über alle Hierarchiestufen. Dabei werden für alle gefundenen Elemente die referenzierten Anwenderfunktionen aufgerufen. Soll ein Funktionstyp nicht aufgerufen werden, so ist für den entsprechenden Parameter `NULL` anzugeben (Definition der referenzierten Anwenderfunktionen siehe **cap_scanall**). Der Rückgabewert der Funktion **cap_scanfelem** ist ungleich Null, wenn ungültige Parameter angegeben wurden, oder wenn eine der referenzierten Anwenderfunktionen einen Fehler an die Funktion **cap_scanfelem** zurückgemeldet hat.

Siehe auch

Funktionen **cap_maccoords**, **cap_scanall**, **cap_scanpool**.

cap_scanpool - Schaltplan Scan über Poolelement (CAP)

Synopsis

```
int cap_scanpool(               // Scan Status
      void;                     // Poolelement
      double;                   // X-Offset (STD2)
      double;                   // Y-Offset (STD2)
      double;                   // Drehwinkel (STD3)
      int [0,1];                // Element in Arbeitsbereich Flag (STD10)
      * int;                    // Makrofunktion
      * int;                    // Verbindungssegmentfunktion
      * int;                    // Polygonfunktion
      * int;                    // Textfunktion
      );
```

Beschreibung

Die Funktion **cap_scanpool** scannt das angegebene Poolelement über alle Hierarchiestufen. Dabei werden für alle gefundenen Elemente die referenzierten Anwenderfunktionen aufgerufen. Soll ein Funktionstyp nicht aufgerufen werden, so ist für den entsprechenden Parameter **NULL** anzugeben (Definition der referenzierten Anwenderfunktionen siehe **cap_scanall**). Der Rückgabewert der Funktion **cap_scanpool** ist ungleich Null, wenn ungültige Parameter angegeben wurden, oder wenn eine der referenzierten Anwenderfunktionen einen Fehler an die Funktion **cap_scanpool** zurückgemeldet hat.

Siehe auch

Funktionen **cap_maccoords**, **cap_scanall**, **cap_scanfelem**.

cap_vecttext - Schaltplan Text vektorisieren (CAP)

Synopsis

```
int cap_vecttext(               // Status
      double;                   // X-Koordinate (STD2)
      double;                   // Y-Koordinate (STD2)
      double;                   // Drehwinkel (STD3)
      int [0,1];                // Spiegelung (STD14)
      double ]0.0,[;            // Text Größe (STD2)
      int [0,[;                 // Text Stil (CAP7)
      string;                   // Text Zeichenkette
      * int;                    // Vektorisierungsfunktion
      );
```

Beschreibung

Die Funktion **cap_vecttext** vektorisiert den übergebenen Text unter Verwendung des aktuell geladenen Zeichensatzes. Dazu wird für jedes Textsegment die übergebene Vektorisierungsfunktion aufgerufen. Der Rückgabewert dieser Funktion ist ungleich Null, wenn ungültige Parameter angegeben wurden oder die vom Benutzer definierte Vektorisierungsfunktion einen Wert ungleich Null zurückgegeben hat.

Vektorisierungsfunktion

```
int vecfuncname(
      double x1,                // X-Koordinate erster Punkt (STD2)
      double y1,                // Y-Koordinate erster Punkt (STD2)
      double x2,                // X-Koordinate zweiter Punkt (STD2)
      double y2                 // Y-Koordinate zweiter Punkt (STD2)
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Der Rückgabewert dieser Funktion sollte ungleich Null sein, wenn ein Fehler aufgetreten ist.

C.3.2 Schaltplaneditor-Funktionen

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp SCM zugeordnet, d.h. diese Funktionen können im **Schaltplaneditor** aufgerufen werden:

scm_askrefname - SCM Referenznamensabfrage (SCM)

Synopsis

```
int scm_askrefname(            // Returns status
      & string;                // Returns reference name
      );
```

Beschreibung

Die Funktion **scm_askrefname** aktiviert einen Dialog zur Referenznamensabfrage, d.h. zur Auswahl von Symbolen auf Stromlaufebene bzw. zur Auswahl von Pins auf Symbolebene. Der Funktionsrückgabewert ist Null bei erfolgreicher Referenzauswahl oder ungleich Null im Fehlerfall.

scm_asktreename - SCM Netznamensabfrage (SCM)

Synopsis

```
int scm_asktreename(           // Status
      & string;                // Rückgabe Netzname
      );
```

Beschreibung

Die Funktion **scm_asktreename** aktiviert einen Dialog zur Netznamensabfrage. Der Funktionsrückgabewert ist Null bei erfolgreicher Netzauswahl oder ungleich Null im Fehlerfall.

scm_attachtextpos - Textverschiebung an SCM-Element anfügen (SCM)

Synopsis

```
int scm_attachtextpos(         // Status
      index C_FIGURE;          // SCM-Element
      string;                  // Text
      double;                  // Text-X-Koordinate (STD2)
      double;                  // Text-Y-Koordinate (STD2)
      double;                  // Textdrehwinkel (STD3)
      double ]0.0,[;           // Textgröße (STD2)
      int [-1,1];              // Textspiegelungsmodus (STD14) oder (-1) zum
Rüsetzen
      );
```

Beschreibung

Die Funktion **scm_attachtextpos** weist die Parameter für Position, Drehwinkel, Größe und Spiegelung an den angegebenen Text des spezifierten SCM-Elements zu. Der Funktionsrückgabewert ist Null bei erfolgreicher Zuweisung, (-1) bei ungültigen Parametern oder (-2) wenn das Layoutelement keine Definition für den angegebenen Text enthält.

Siehe auch

Funktion **scm_storetext**.

scm_checkbustapplot - SCM-Busanschluß Plotstatus abfragen (SCM)

Synopsis

```
int scm_checkbustapplot(       // Rückgabe Plotstatus
      index C_FIGURE;          // Busverbindungselement
      index C_BUSTAP;          // Busanschluß
      );
```

Beschreibung

Die Funktion **scm_checkbustapplot** gibt den Wert 1 zurück wenn die Plotausgabe für den sepzifizierten Busanschluß deaktiviert ist. Im anderen Fall ergibt sich der Rückgabewert zu Null.

scm_checkjunctplot - SCM-Verbindungspunktmarker Plotstatus abfragen (SCM)

Synopsis

```
int scm_checkjunctplot(       // Plotstatus
      double;                 // Verbindungspunkt X-Position (STD2)
      double;                 // Verbindungspunkt Y-Position (STD2)
      );
```

Beschreibung

Die Funktion **scm_checkjunctplot** überprüft, ob eine Plotausgabe von Verbindungspunktmarkern an der angegebenen Koordinate durchzuführen ist (Funktionsrückgabewert 0) oder unterdrückt werden soll (Funktionsrückgabewert 1).

scm_chkattrname - SCM-Attributname validieren (SCM)

Synopsis

```
int scm_chkattrname(          // Rückgabe ungleich Null bei ungültigem
Attributnamen
      string;                 // Attributname
      );
```

Beschreibung

Die Funktion **scm_chkattrname** überprüft ob der angegebene Attributname gültig ist und eine Attributwertzuweisung zulässt. Der Funktionsrückgabewert ist Null bei gültigem Attributnamen bzw. ungleich Null wenn der Attributname ungültig ist.

Siehe auch

Funktion **scm_setpartattrib**.

scm_conseggrpchg - SCM Verbindungssegment Gruppenflag ändern (SCM)

Synopsis

```
int scm_conseggrpchg(         // Status
      index C_CONSEG;         // Verbindungssegment
      int [0,6];              // Neue Gruppenzugehörigkeit (STD13)
                              //    |4 - Anzeige Gruppenstatusmeldung
      );
```

Beschreibung

Die Funktion **scm_conseggrpchg** ändert die Gruppenzugehörigkeit des übergebenen Verbindungssegments. Der Rückgabewert ist Null bei erfolgter Änderung oder (-1) wenn das übergebene Element ungültig ist.

scm_deflibname - SCM Setup default Bibliothek (SCM)

Synopsis

```
string scm_deflibname(        // Default SCM Bibliotheksname
      );
```

Beschreibung

Der Rückgabewert der Funktion **scm_deflibname** entspricht dem in der Setupdatei eingestellten Defaultnamen für die SCM-Bibliothek.

scm_defloglname - SCM Setup default Packager Bibliothek (SCM)

Synopsis

```
string scm_defloglname(       // Default Packager Bibliotheksname
      );
```

Beschreibung

Der Rückgabewert der Funktion **scm_defloglname** entspricht dem in der Setupdatei eingestellten Defaultnamen für die vom **Packager** verwendete Layoutbibliothek. Dieser Dateiname wird üblicherweise bei der Abfrage von logischen Bauteildefinitionen.

Warnung

Funktion **con_getlogpart**; Utilityprogramm **BSETUP**.

scm_defsegbus - SCM Verbindungssegment Busdefinition (SCM)

Synopsis

```
int scm_defsegbus(            // Status
      & index C_CONSEG;       // Verbindungssegment
      );
```

Beschreibung

Die Funktion **scm_defsegbus** definiert das übergebene Verbindungssegment und alle damit verbundenen Segmente zum Bus. Der Rückgabewert ist ungleich Null, wenn die Busdefinition nicht durchgeführt werden konnte.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **C_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

scm_delconseg - SCM Verbindungssegment löschen (SCM)

Synopsis

```
int scm_delconseg(            // Status
      & index C_CONSEG;       // Verbindungssegment
      );
```

Beschreibung

Die Funktion **scm_delconseg** löscht das übergebene Verbindungssegment. Der Rückgabewert ist Null bei erfolgter Löschung und (-1), wenn das übergebene Element ungültig ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **C_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

scm_delelem - SCM Element löschen (SCM)

Synopsis

```
int scm_delelem(                // Status
      & index C_FIGURE;         // Element
      );
```

Beschreibung

Die Funktion **scm_delelem** löscht das übergebene Element aus der Elementliste. Der Rückgabewert ist Null bei erfolgter Löschung und (-1), wenn das übergebene Element ungültig ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **C_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

Siehe auch

Funktion **scm_drawelem**.

scm_drawelem - SCM Elementanzeige aktualisieren (SCM)

Synopsis

```
void scm_drawelem(
      index C_FIGURE;           // Element
      int [0, 4];               // Zeichenmodus (STD19)
      );
```

Beschreibung

Die Funktion **scm_drawelem** aktualisiert die Anzeige des angegebenen Elements unter Verwendung des spezifizierten Zeichenmodus.

Siehe auch

Funktion **scm_delelem**.

scm_elemangchg - SCM Elementwinkel ändern (SCM)

Synopsis

```
int scm_elemangchg(             // Status
      & index C_FIGURE;         // Element
      double;                   // Neuer Winkel (STD3)
      );
```

Beschreibung

Die Funktion **scm_elemangchg** ändert den Drehwinkel des übergebenen Elements. Der Drehwinkel wird ausgehend vom Nullwinkel eingestellt, d.h. der vorhergehende Drehwinkel des Elements hat keinen Einfluss auf das Ergebnis. Die Winkelangabe wird als Bogenmaßwert interpretiert. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht drehbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **C_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

scm_elemgrpchg - SCM Element Gruppenflag ändern (SCM)

Synopsis

```
int scm_elemgrpchg(            // Status
      index C_FIGURE;          // Element
      int [0,6];               // Neue Gruppenzugehörigkeit (STD13)
                               //    |4 - Anzeige Gruppenstatusmeldung
      );
```

Beschreibung

Die Funktion **scm_elemgrpchg** ändert die Gruppenzugehörigkeit des übergebenen Elements. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig ist oder (-2) wenn es sich nicht um ein gruppenselektierbares Element handelt. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

scm_elemmirrchg - SCM Elementspiegelung ändern (SCM)

Synopsis

```
int scm_elemmirrchg(           // Status
      & index C_FIGURE;        // Element
      int [0,1];               // Neuer Spiegelungsmodus (STD14)
      );
```

Beschreibung

Die Funktion **scm_elemmirrchg** ändert den Spiegelungsmodus des übergebenen Elements. Der Spiegelungsmodus kann bei Texten und Referenzen geändert werden. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig ist oder (-2) wenn es nicht gespiegelt werden kann. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **C_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

scm_elemposchg - SCM Elementposition ändern (SCM)

Synopsis

```
int scm_elemposchg(            // Status
      & index C_FIGURE;        // Element
      double;                  // X-Position (STD2)
      double;                  // Y-Position (STD2)
      );
```

Beschreibung

Die Funktion **scm_elemposchg** ändert die Position des übergebenen Elements. Bei Polygonen wird das Polygon so verschoben, dass der erste Punkt des Polygons auf der angegebenen Position zu liegen kommt. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht positionierbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **C_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

scm_elemsizechg - SCM Elementgröße ändern (SCM)

Synopsis

```
int scm_elemsizechg(          // Status
      & index C_FIGURE;       // Element
      double;                 // Neue Größe (STD2)
      );
```

Beschreibung

Die Funktion **scm_elemsizechg** ändert die Größe des übergebenen Elements. Eine Größenänderung ist nur bei Textelementen möglich. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht größenveränderbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **C_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

scm_findpartplc - Layoutbauteil Platzierungsstatus abfragen (BAE HighEnd) (SCM)

Synopsis

```
int scm_findpartplc(          // Platzierungsstatus
      string;                 // Bauteilname
      );
```

Beschreibung

Die Funktion **scm_findpartplc** kann im **BAE HighEnd Schaltplaneditor** dazu benutzt werden, den Platzierungsstatus von Layoutbauteilen zu prüfen. Der Funktionsrückgabewert ist 1 wenn das Bauteil mit dem angegebenen Bauteilnamen auf dem Projektlayout platziert ist. Andernfalls wird der Wert Null zurückgegeben.

scm_getdblpar - SCM Doubleparameter abfragen (SCM)

Synopsis

```
int scm_getdblpar(            // Status
      int [0,[;               // Parametertyp/-nummer:
                              //    0 = Plot Skalierungsfaktor
                              //    1 = Plotter HPGL-Geschwindigkeit
                              //    2 = Plotter Stiftbreite (STD2)
                              //    3 = X-Koord. letzte Gruppenplatzierung (STD2)
                              //    4 = Y-Koord. letzte Gruppenplatzierung (STD2)
                              //    5 = Standardsymbolplatzierungswinkel (STD3)
                              //    6 = Standardtextgröße (STD2)
                              //    7 = Standardtextplatzierungswinkel (STD3)
      & double;               // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **scm_getdblpar** dient der Abfrage von mit **scm_setdblpar** im **Schaltplaneditor** gesetzten Parametern vom Typ `double`. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **scm_getintpar**, **scm_getstrpar**, **scm_setdblpar**, **scm_setintpar**, **scm_setstrpar**.

scm_getgroupdata - SCM Gruppenplatzierungsdaten abfragen (SCM)

Synopsis

```
int scm_getgroupdata(          // Status
      & double;                // Gruppenbezugspunkt X-Koordinate (STD2)
      & double;                // Gruppenbezugspunkt Y-Koordinate (STD2)
      & double;                // Gruppendrehwinkel (STD3)
      & double;                // Gruppenskalierungsfaktor
      & int;                   // Gruppenspiegelungsmodus
      & double;                // Gruppenquadrant X-Koordinate (STD2)
      & double;                // Gruppenquadrant Y-Koordinate (STD2)
      & int;                   // Gruppenquadrantmodus
      & int;                   // Gruppenbereichsmodus
      );
```

Beschreibung

Die Funktion **scm_getgroupdata** ermöglicht die Abfrage der aktuellen Eingabedaten während interaktiver Gruppenplatzierungen im **Schaltplaneditor**. Der Funktionsrückgabewert ist ungleich Null wenn gerade keine Gruppenplatzierungsinteraktion aktiviert ist.

Siehe auch

Funktion **scm_getinputdata**.

scm_gethighlnet - SCM Netz Highlightmodus abfragen (SCM)

Synopsis

```
int scm_gethighlnet(           // Status
      int [-1,[;               // Netznummer oder -1 für Highlightfokusmodusabfrage
      & int;                   // Highlightmodus
      );
```

Beschreibung

Mit der Funktion **scm_gethighlnet** kann der Highlightmodus für das Netz mit der angegebenen Netznummer abgefragt werden. Bei aktiviertem Netzhighlight wird im Parameter für den Highlightmodus ein Wert ungleich Null zurückgegegen, bei deaktiviertem Netzhighlight ergibt sich der Highlightmodusparameter zu Null. Der Funktionsrückgabewert ergibt sich zu einem Wert ungleich Null, wenn die Abfrage erfolgreich war, andernfalls (Netz nicht gefunden, ungültige Parameter) wird der Wert Null zurückgegeben.

Siehe auch
Funktion **scm_highlnet**.

scm_gethpglparam - SCM HP-GL-Parameter abfragen (SCM)

Synopsis

```
void scm_gethpglparam(
      & string;              // HP-GL Plotdateiname
      & double;              // HP-GL Maßstab
      & double;              // HP-GL Geschwindigkeit (-1.0 volle Geschw.)
      & double;              // HP-GL Stiftbreite (STD2)
      & int;                 // HP-GL Füllmodus:
                             //    0 = Füllen aus
                             //    1 = Füllen ein
      & int;                 // HP-GL Rotationsmodus:
                             //        0 = keine Drehung
                             //        1 = Drehung um 90 Grad
                             //    sonst = automatische Drehung
      );
```

Beschreibung

Die Funktion **scm_gethpglparam** gibt die aktuell im **Schaltplaneditor** eingestellten HP-GL-Plotparameter zurück.

scm_getinputdata - SCM Eingabedaten abfragen (SCM)

Synopsis

```
int scm_getinputdata(        // Status
      & double;              // Ursprüngliche X-Koordinate (STD2)
      & double;              // Ursprüngliche Y-Koordinate (STD2)
      & double;              // Ursprüngliche Breite (STD2)
      & int;                 // Ursprünglicher Anzeigeelementtyp (SCM1)
      & double;              // Aktuelle X-Koordinate (STD2)
      & double;              // Aktuelle Y-Koordinate (STD2)
      & double;              // Aktuelle Breite (STD2)
      & int;                 // Aktueller Anzeigelementtyp (SCM1)
      & void;                // Eingabemodus/Element
      & double;              // Erstes Segment Start-X-Koordinate (STD2) */
      & double;              // Erstes Segment Start-Y-Koordinate (STD2) */
      & double;              // Erster Kreismittelpunkt X-Koordinate (STD2) */
      & double;              // Erster Kreismittelpunkt Y-Koordinate (STD2) */
      & int;                 // Erster Kreismittelpunkt Typ (STD15) */
      & double;              // Letztes Segment Start-X-Koordinate (STD2) */
      & double;              // Letztes Segment Start-Y-Koordinate (STD2) */
      & double;              // Letzter Kreismittelpunkt X-Koordinate (STD2) */
      & double;              // Letzter Kreismittelpunkt Y-Koordinate (STD2) */
      & int;                 // Letzter Kreismittelpunkt Typ (STD15) */
      );
```

Beschreibung

Die Funktion **scm_getinputdata** ermöglicht die Abfrage der aktuellen Eingabedaten während interaktiver Platzierungen im **Schaltplaneditor**. Die Interpretation der Platzierungsdaten ist entsprechend dem zurückgelieferten Funktionsparameter für den Eingabemodus bzw. das Platzierungselement vorzunehmen. Der Funktionsrückgabewert ist ungleich Null wenn gerade keine Platzierungsinteraktion aktiviert ist.

Siehe auch
Funktion **scm_getgroupdata**.

scm_getintpar - SCM Integerparameter abfragen (SCM)

Synopsis

```
int scm_getintpar(              // Status
      int [0,[;                 // Parametertyp/-nummer:
                                //    0 = Pickpunktanzeigemodus:
                                //        0 = keine Pickpunktanzeige
                                //        1 = Pickpunktanzeige
                                //    1 = Symbol/Gruppen-Reroutingmodus:
                                //        Bit 0/1: Routermodus
                                //          0 = Router inaktiv
                                //          1 = Symbol- & Gruppenrouting
                                //          2 = Nur Symbolrouting
                                //          3 = Nur Gruppenrouting
                                //        Bit 2: Quickshot-Routing Flag
                                //    2 = Symbolbewegungsmodus:
                                //        0 = Symboltextoffsets zurücksetzen
                                //        1 = Symboltextoffsets beibehalten
                                //    3 = Typ der zuletzt platzierten Referenz:
                                //        (-1) = keine Referenz platziert
                                //        0 = Symbol
                                //        1 = Label
                                //        2 = Modulport
                                //    4 = Warnungen bei Verbindung benannter Netze:
                                //        0 = Warnmeldungsanzeige deaktiviert
                                //        1 = Warnmeldungsanzeige aktiviert
                                //    5 = Elementpickmodus:
                                //        0 = Bester Pick
                                //        1 = Pickelementauswahl
                                //    6 = Farbmodus Generischer Drucker:
                                //        0 = schwarzweiss
                                //        1 = farbig
                                //    7 = Warnmeldungsmodus:
                                //        Bit 0: $-Umbenennungen unterdrücken
                                //        Bit 1: Netzzusammenfassungen unterdrücken
                                //        Bit 2: Modulportwarnungen unterdrückt
                                //        Bit 4: Gruppensymbolumbenennungswarnungen
                                //                unterdrückt
                                //        Bit 5: Variantenvergleichswarnungen
                                //                unterdrückt
                                //    8 = Eingabemodus für Namen:
                                //        0 = Autopattern für Symbolnamen
                                //        1 = Eingabeaufforderung für Symbolnamen
                                //    9 = Infoanzeigeflag:
                                //        0 = keine Infoanzeige
                                //        1 = automatische Infoanzeige
                                //    10 = Infoanzeigemodus:
                                //        0 = keine automatische Infoanzeige
                                //        1 = vollständige Infoanzeige
                                //        2 = nur netzspezifische Infoanzeige
                                //    11 = Labelreroutingmodus:
                                //        0 = kein automatisches Labelrerouting
                                //        1 = automatisches Labelrerouting
                                //    12 = Offset Subsymbolnummer
                                //    13 = Generische Plotausgabe Skalierungsmodus:
                                //        0 = fester Skalierungsfaktor
                                //        1 = automatische Skalierung auf Seite
                                //    14 = Generische Plotausgabe Farbmodus:
                                //        0 = schwarz/weiss
                                //        1 = aktuelle Farbeinstellung benutzen
                                //    15 = HPGL Füllmodus:
                                //        0 = Konturen zeichnen
                                //        1 = Flächen füllen
                                //        2 = Flächen füllen, Linien/Texte zeichnen
                                //    16 = Flächenpolygoneditiermodus:
                                //        0 = keine geschlossenen Linienzüge
                                //        1 = immer geschlossene Linienzüge
                                //        2 = Abfrage zum Schließen von Linienzügen
```

```
                          //    17 = Gruppenreroutingmodus:
                          //        0 = kein Löschen von Außenantennen
                          //        1 = Löschen erstes Außenantennensegment
                          //        2 = Löschen alle Außenantennensegmente
                          //    18 = Benutzerspezifisches
                          //          Standardeinheitensystem:
                          //        0 = metrisch
                          //        1 = zöllig
                          //    19 = Plotvorschaumodus:
                          //        0 = keine Plotvorschau
                          //        1 = Plotterstiftbreite
                          //    20 = Autosave Intervall
                          //    21 = Automatische Verbindungsecken
                          //    22 = Winkelfreigabeflag
                          //    23 = Standardsymbolspiegelung
                          //    24 = Gruppenbewegtdarstellung:
                          //        0 = Bewegtbild ein
                          //        1 = Bewegtbild alles
                          //        2 = Bewegtbild kontinuierlich
                          //    25 = Zwischenablagetext-
                          //          Platzierungsanforderung
                          //    26 = Signalrouter Routingbereich
                          //    27 = Automatische Busanzapfung
                          //    28 = Signalrouter Markerscan
                          //    29 = Segmentbewegungsmodus:
                          //        0 = Bewegen ohne Nachbarn
                          //        1 = Bewegen mit Nachbarn
                          //        2 = Nachbarn anpassen
                          //        |4 = Endpunkt folgt Segment
                          //    30 = Gruppenwinkelfreigabemodus:
                          //        0 = Gruppenwinkel einhalten
                          //        1 = Gruppenwinkel automatisch freigeben
                          //    31 = Standard-Textspiegelungsmodus (STD14)
                          //    32 = Standard-Textmodus (CAP1|CAP7)
                          //    33 = Symbol $noplc Plotsichtbarkeitsbeziehung:
                          //        Bit 0 = $noplc setzen bei
                          //                 Plotsichtbarkeitsänderung
                          //        Bit 1 = Plotsichtbarkeit setzen bei
                          //                 $noplc-Änderung
                          //    34 = Netzplanlisting Maximallänge [ 3,200]
                          //    35 = Flag - Einzeleckenbearbeitung
                          //    36 = Flag - Unroutes-Liniengenerierung
                          //    37 = Flag - Polygonbearbeitung
                          //          Autocomplete-Modus
                          //    38 = Fehlerhighlightmodus:
                          //        0 = Fehlerhighlight
                          //        1 = Fehlermuster/-strichelung
                          //    39 = Flag - Automirror horizontal bus taps
                          //    40 = Symbol-/Label-Tagmodus:
                          //        0 = Standardsymbol
                          //        1 = Virtueller Tag
                          //        2 = Netzlistentag
                          //    41 = Verbindungstrennmodus:
                          //        0 = Verbindungen nicht trennen
                          //        1 = Verbindungen an Zweipinsymbolen
                          //             trennen
                          //        2 = Verbindungen trennen
                          //    42 = Symbolverbindungstrennmodus:
                          //        0 = Verbindungen nicht trennen
                          //        1 = Verbindungen an Zweipinsymbolen
                          //             trennen
                          //        2 = Verbindungen trennen
                          //    43 = Von scm_checkjunctplot gesetzte
                          //          Verbindungspunktmarker-
                          //          Gruppensegmentanzahl
                          //    44 = Aktuelle Anzahl Projektpläne
                          //    45 = Anzeigeklassenbits Verbindung (SCM2)
                          //    46 = Anzeigeklassenbits Bus (SCM2)
                          //    47 = Anzeigeklassenbits Text (SCM2)
                          //    48 = Anzeigeklassenbits Kommentartext (SCM2)
                          //    49 = Anzeigeklassenbits Grafikfläche (SCM2)
```

```
                              //    50 = Anzeigeklassenbits Grafiklinie (SCM2)
                              //    51 = Anzeigeklassenbits Strichlinie (SCM2)
                              //    52 = Anzeigeklassenbits Verbindungsfläche
                              //          (SCM2)
                              //    53 = Anzeigeklassenbits Netzbereich (SCM2)
                              //    54 = Anzeigeklassenbits Makroumrandung (SCM2)
                              //    55 = Anzeigeklassenbits Tag (SCM2)
      & int;                  // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **scm_getintpar** dient der Abfrage von mit **scm_setintpar** im **Schaltplaneditor** gesetzten Integerparametern. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **scm_getdblpar**, **scm_getstrpar**, **scm_setdblpar**, **scm_setintpar**, **scm_setstrpar**.

scm_getstrpar - SCM Stringparameter abfragen (SCM)

Synopsis

```
int scm_getstrpar(            // Status
      int [0,[;               // Parametertyp/-nummer:
                              //    0 = Name zuletzt platziertes
                              //         Bibliothekselement
                              //    1 = Name zuletzt platziertes benanntes Element
                              //    2 = Name zuletzt platziertes Netz
                              //    3 = Name zuletzt platzierter Busanschluß
                              //    4 = Inhalt zuletzt platzierte Zeichenkette
                              //    5 = Symbolnamensmuster
                              //    6 = Nächster platzierter Text
                              //    7 = Bibliothek zuletzt platziertes Makro
                              //    8 = Fehlerhafte Busanschlussbezeichnung
                              //    9 = Fehlerhafte Busbezeichnung
                              //    10 = Nächster freier Name
                              //    11 = Current hierachical block reference name
                              //    12 = Last picked attribute name
                              //    13 = Last picked attribute value
                              //    14 = Autosave path name
      & string;               // Returns parameter value
      );
```

Beschreibung

Die Funktion **scm_getstrpar** dient der Abfrage von im **Schaltplaneditor** gesetzten Stringparametern. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch
Funktionen **scm_getdblpar**, **scm_getintpar**, **scm_setdblpar**, **scm_setintpar**, **scm_setstrpar**.

scm_highlnet - SCM Netz Highlightmodus setzen (SCM)

Synopsis

```
int scm_highlnet(               // Status
      int [-1,[;                // Netznummer
      int [0,1];                // Highlightmodus
      );
```

Beschreibung

Mit der Funktion **scm_highlnet** kann der Highlightmodus für das Netz mit der angegebenen Netznummer gesetzt werden. Zur Aktivierung des Netzhighlights ist das Bit 1 der Highlighmodusparameter auf 1 zu setzen, die Deaktivierung erfolgt entsprechend durch Zuweisung von Null. Der Funktionsrückgabewert ergibt sich zu einem Wert ungleich Null bei erfolgreicher Änderung des netzspezifischen Highlightmodus, andernfalls (Netz nicht gefunden, ungültige Parameter) wird der Wert Null zurückgegeben.

Siehe auch

Funktion **scm_gethighlnet**.

scm_pickanyelem - Beliebiges SCM Element selektieren (SCM)

Synopsis

```
int scm_pickanyelem(            // Status
      & index C_FIGURE;         // Rückgabe selektiertes Anzeigeelement
      & index C_CONSEG;         // Rückgabe selektiertes Verbindungssegment
      & index C_BUSTAP;         // Rückgabe selektierter Busanschluß
      & int;                    // Rückgabe selektierter Elementtyp:
                                //    0 = Anzeigeelement
                                //    1 = Verbindungssegment
                                //    2 = Busanschluß
      int;                      // Pickelementtypmenge ((CAP3 außer 7)<<1 verodert)
      );
```

Beschreibung

Die Funktion **scm_pickanyelem** aktiviert eine Mausinteraktion zur Selektion eines Elements aus der angegebenen Pickelementtypmenge. Das selektierte Element wird in einem der ersten drei Parameter zurückgegeben. Der Rückgabewert für den selektierten Elementtyp kann zur Bestimmung des gültigen Parameters für das selektierte Element ausgewertet werden. Der Funktionsrückgabewert ist Null bei erfolgter Selektion und ungleich Null wenn an der Pickposition kein Element gefunden wurde.

Siehe auch

Funktionen **scm_pickbustap**, **scm_pickconseg**, **scm_pickelem**, **scm_setpickconseg**.

scm_pickbustap - SCM Bustap selektieren (SCM)

Synopsis

```
int scm_pickbustap(             // Status
      & index C_BUSTAP;         // Rückgabe selektiertes Busanschlusselement
      );
```

Beschreibung

Mit der Funktion **scm_pickbustap** kann vom Benutzer mit der Maus ein Busanschluss selektiert werden. Der Rückgabewert ist Null bei erfolgter Selektion und (-1) wenn an der Pickposition kein Busansachlusselement gefunden wurde.

Siehe auch

Funktionen **scm_pickanyelem**, **scm_pickconseg**, **scm_pickelem**, **scm_setpickconseg**.

scm_pickconseg - SCM Verbindungssegment selektieren (SCM)

Synopsis

```
int scm_pickconseg(            // Status
      & index C_CONSEG;        // Rückgabe Verbindungssegment
      );
```

Beschreibung

Mit der Funktion **scm_pickconseg** kann vom Benutzer mit der Maus ein Verbindungssegment selektiert werden. Der Rückgabewert ist Null bei erfolgter Selektion und (-1) wenn an der Pickposition kein Verbindungssegment gefunden wurde.

Siehe auch

Funktionen **scm_pickanyelem**, **scm_pickbustap**, **scm_pickelem**, **scm_setpickconseg**.

scm_pickelem - SCM Element selektieren (SCM)

Synopsis

```
int scm_pickelem(              // Status
      & index C_FIGURE;        // Rückgabe selektiertes Element
      int [1,11];              // Elementtyp (CAP3 außer 2 und 7)
      );
```

Beschreibung

Mit der Funktion **scm_pickelem** kann vom Benutzer mit der Maus ein Element des gewünschten übergebenen Typs selektiert werden, wobei mit dieser Funktion keine Verbindungen selektiert werden können. Der Rückgabewert ist Null bei erfolgter Selektion und (-1) wenn an der Pickposition kein Element des gewünschten Typs gefunden wurde.

Siehe auch

Funktionen **scm_pickanyelem**, **scm_pickbustap**, **scm_pickconseg**, **scm_setpickconseg**.

scm_setdblpar - SCM Doubleparameter setzen (SCM)

Synopsis

```
int scm_setdblpar(             // Status
      int [0,[;                // Parametertyp/-nummer:
                               //    0 = Plot Skalierungsfaktor
                               //    1 = Plotter HPGL-Geschwindigkeit
                               //    2 = Plotter Stiftbreite (STD2)
                               //    3 = X-Koordinate letzte
                               //          Gruppenplatzierung (STD2)
                               //    4 = Y-Koordinate letzte
                               //          Gruppenplatzierung (STD2)
                               //    5 = Standardsymbolplatzierungswinkel (STD3)
                               //    6 = Standardtextgröße (STD2)
                               //    7 = Standardtextplatzierungswinkel (STD3)
      double;                  // Parameterwert
      );
```

Beschreibung

Die Funktion **scm_setdblpar** dient dazu, Systemparameter vom Typ `double` im **Schaltplaneditor** zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **scm_setdblpar** gesetzten Systemparametern können mit der Funktion **scm_getdblpar** abgefragt werden.

Siehe auch

Funktionen **scm_getdblpar**, **scm_getintpar**, **scm_getstrpar**, **scm_setintpar**, **scm_setstrpar**.

scm_setintpar - SCM Integerparameter setzen (SCM)

Synopsis

```
int scm_setintpar(              // Status
      int [0,[;                 // Parametertyp/-nummer:
                                //    0 = Pickpunktanzeigemodus:
                                //        0 = keine Pickpunktanzeige
                                //        1 = Pickpunktanzeige
                                //    1 = Symbol/Gruppen-Reroutingmodus:
                                //        Bit 0/1: Routermodus
                                //          0 = Router inaktiv
                                //          1 = Symbol- & Gruppenrouting
                                //          2 = Nur Symbolrouting
                                //          3 = Nur Gruppenrouting
                                //        Bit 2: Quickshot-Routing Flag
                                //    2 = Symbolbewegungsmodus:
                                //        0 = Symboltextoffsets zurücksetzen
                                //    3 = Typ der zuletzt platzierten Referenz:
                                //        Parameter nur für Lesezugriff!
                                //    4 = Warnungen bei Verbindung benannter Netze:
                                //        0 = Warnmeldungsanzeige deaktivieren
                                //        1 = Warnmeldungsanzeige aktivieren
                                //    5 = Elementpickmodus:
                                //        0 = Bester Pick
                                //        1 = Pickelementauswahl
                                //    6 = Farbmodus Generischer Drucker:
                                //        0 = schwarzweiss
                                //        1 = farbig
                                //    7 = Warnmeldungsmodus:
                                //        Bit 0 = $-Umbenennungen unterdrücken
                                //        Bit 1: Netzzusammenfassungen unterdrücken
                                //        Bit 2: Modulportwarnungen unterdrücken
                                //        Bit 4: Gruppensymbolumbenennungswarnungen
                                //               unterdrücken
                                //        Bit 5: Variantenvergleichswarnungen
                                //               unterdrücken
                                //    8 = Eingabemodus für Namen:
                                //        0 = Autopattern für Symbolnamen
                                //        1 = Eingabeaufforderung für Symbolnamen
                                //    9 = Infoanzeigeflag:
                                //        0 = keine Infoanzeige
                                //        1 = automatische Infoanzeige
                                //    10 = Infoanzeigemodus:
                                //        0 = keine automatische Infoanzeige
                                //        1 = vollständige Infoanzeige
                                //        2 = nur netzspezifische Infoanzeige
                                //    11 = Labelreroutingmodus:
                                //        0 = kein automatisches Labelrerouting
                                //        1 = automatisches Labelrerouting
                                //    12 = Offset Subsymbolnummer
                                //    13 = Generische Plotausgabe Skalierungsmodus:
                                //        0 = fester Skalierungsfaktor
                                //        1 = automatische Skalierung auf Seite
                                //    14 = Generische Plotausgabe Farbmodus:
                                //        0 = schwarz/weiss
                                //        1 = aktuelle Farbeinstellung benutzen
                                //    15 = HPGL Füllmodus:
                                //        0 = Konturen zeichnen
                                //        1 = Flächen füllen
                                //        2 = Flächen füllen, Linien/Texte zeichnen
                                //    16 = Flächenpolygoneditiermodus:
                                //        0 = keine geschlossenen Linienzüge
                                //        1 = immer geschlossene Linienzüge
                                //        2 = Abfrage zum Schließen von Linienzügen
```

```
                                //    17 = Gruppenreroutingmodus:
                                //        0 = kein Löschen von Außenantennen
                                //        1 = Löschen erstes Außenantennensegment
                                //        2 = Löschen alle Außenantennensegmente
                                //    18 = Benutzerspezifisches
                                //          Standardeinheitensystem:
                                //        0 = metrisch
                                //        1 = zöllig
                                //    19 = Plotvorschaumodus:
                                //        0 = keine Plotvorschau
                                //        1 = Plotterstiftbreite
                                //    20 = Autosave Intervall
                                //    21 = Automatische Verbindungsecken
                                //    22 = Winkelfreigabeflag
                                //    23 = Standardsymbolspiegelung
                                //    24 = Gruppenbewegtdarstellung:
                                //        0 = Bewegtbild ein
                                //        1 = Bewegtbild alles
                                //        2 = Bewegtbild kontinuierlich
                                //    25 = Zwischenablagetext-
                                //          Platzierungsanforderung
                                //    26 = Signalrouter Routingbereich
                                //    27 = Automatische Busanzapfung
                                //    28 = Signalrouter Markerscan
                                //    29 = Segmentbewegungsmodus:
                                //        0 = Bewegen ohne Nachbarn
                                //        1 = Bewegen mit Nachbarn
                                //        2 = Nachbarn anpassen
                                //        |4 = Endpunkt folgt Segment
                                //    30 = Gruppenwinkelfreigabemodus:
                                //        0 = Gruppenwinkel einhalten
                                //        1 = Gruppenwinkel automatisch freigeben
                                //    31 = Standard-Textspiegelungsmodus (STD14)
                                //    32 = Standard-Textmodus (CAP1|CAP7)
                                //    33 = Symbol $noplc Plotsichtbarkeitsbeziehung:
                                //        Bit 0 = $noplc setzen bei
                                //                 Plotsichtbarkeitsänderung
                                //        Bit 1 = Plotsichtbarkeit setzen bei
                                //                 $noplc-Änderung
                                //    34 = Netzplanlisting Maximallänge [ 3,200]
                                //    35 = Flag - Einzeleckenbearbeitung
                                //    36 = Flag - Unroutes-Liniengenerierung
                                //    37 = Flag - Polygonbearbeitung
                                //          Autocomplete-Modus
                                //    38 = Fehlerhighlightmodus:
                                //        0 = Fehlerhighlight
                                //        1 = Fehlermuster/-strichelung
                                //    39 = Flag - Automirror horizontal bus taps
                                //  [ 40 = Systemparameter - kein Schreibzugriff ]
                                //    41 = Verbindungstrennmodus:
                                //        0 = Verbindungen nicht trennen
                                //        1 = Verbindungen an Zweipinsymbolen
                                //            trennen
                                //        2 = Verbindungen trennen
                                //    42 = Symbolverbindungstrennmodus:
                                //        0 = Verbindungen nicht trennen
                                //        1 = Verbindungen an Zweipinsymbolen
                                //            trennen
                                //        2 = Verbindungen trennen
                                //  [ 43 = Systemparameter - kein Schreibzugriff ]
                                //  [ 44 = Systemparameter - kein Schreibzugriff ]
                                //    45 = Anzeigeklassenbits Verbindung (SCM2)
                                //    46 = Anzeigeklassenbits Bus (SCM2)
                                //    47 = Anzeigeklassenbits Text (SCM2)
                                //    48 = Anzeigeklassenbits Kommentartext (SCM2)
                                //    49 = Anzeigeklassenbits Grafikfläche (SCM2)
                                //    50 = Anzeigeklassenbits Grafiklinie (SCM2)
                                //    51 = Anzeigeklassenbits Strichlinie (SCM2)
                                //    52 = Anzeigeklassenbits Verbindungsfläche
(SCM2)
                                //    53 = Anzeigeklassenbits Netzbereich (SCM2)
```

```
                              //    54 = Anzeigeklassenbits Makroumrandung (SCM2)
                              //    55 = Anzeigeklassenbits Tag (SCM2)
      int;                    // Parameterwert
      );
```

Beschreibung

Die Funktion **scm_setintpar** dient dazu, Systemparameter vom Typ **int** im **Schaltplaneditor** zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **scm_setintpar** gesetzten Systemparametern können mit der Funktion **scm_getintpar** abgefragt werden.

Siehe auch

Funktionen **scm_getdblpar**, **scm_getintpar**, **scm_getstrpar**, **scm_setdblpar**, **scm_setstrpar**.

scm_setpartattrib - SCM Bauteilattribut setzen (SCM)

Synopsis

```
int scm_setpartattrib(        // Status
      string;                 // Bauteilname
      string;                 // Attributname
      string;                 // Attributwert
      int;                    // Bauteilbearbeitungsflags:
                              //  Bit 0: vom Bildschirm entfernen
                              //  Bit 1: Bildschirmanzeige aktualisieren
                              //  Bit 2: Zuweisung erzwingen, Varianten
deaktivieren
      );
```

Beschreibung

Die Funktion **scm_setpartattrib** setzt für das namentlich spezifizierte Bauteil den Wert des angegebenen Attributs. Die maximal speicherbare Stringlänge für den Attributwert beträgt 40 Zeichen. Der Rückgabewert ist Null bei erfolgreicher Attributwertdefinition, (-1) wenn kein gültiges Element geladen ist, (-2) bei fehlenden bzw. ungültigen Parametern, (-3) wenn das Bauteil nicht gefunden wurde, oder (-4) wenn das Attribut mit dem angegebenen Name nicht am Bauteil definiert ist.

Siehe auch

Funktion **scm_chkattrname**.

scm_setpickconseg - SCM Defaultverbindungspickelement setzen (SCM)

Synopsis

```
int scm_setpickconseg(        // Status
      index C_CONSEG;         // Verbindungssegment
      );
```

Beschreibung

Die Funktion **scm_setpickconseg** selektiert das angegebene Verbindungssegment als Defaultelement für nachfolgende Verbindungssegmentpickoperationen. Der Rückgabewert ist Null bei erfolgreicher Selektion oder (-1) im Fehlerfall.

Siehe auch

Funktionen **scm_pickanyelem**, **scm_pickconseg**, **scm_pickbustap**, **scm_pickelem**.

scm_setpickelem - Set SCM default pick element (SCM)

Synopsis

```
int scm_setpickelem(           // Status
      index C_FIGURE;          // Defaultpickelement
      );
```

Beschreibung

Die Funktion **scm_setpickelem** setzt ein Defaultelement für nachfogende Pickoperationen im **Schaltplaneditor**. Der Funktionsrückgabewert ist ungleich Null im Fehlerfall.

Siehe auch

Funktion **scm_pickelem**.

scm_setstrpar - SCM Stringparameter setzen (SCM)

Synopsis

```
int scm_setstrpar(             // Status
      int [0,[;                // Parametertyp/-nummer:
                               //  [ 0 = Systemparameter schreibgeschützt ]
                               //  [ 1 = Systemparameter schreibgeschützt ]
                               //  [ 2 = Systemparameter schreibgeschützt ]
                               //  [ 3 = Systemparameter schreibgeschützt ]
                               //    4 = Inhalt zuletzt platzierte Zeichenkette
                               //    5 = Symbolnamensmuster
                               //  [ 6 = Systemparameter - kein Schreibzugriff ]
                               //  [ 7 = Systemparameter schreibgeschützt ]
                               //  [ 8 = Systemparameter schreibgeschützt ]
                               //  [ 9 = Systemparameter schreibgeschützt ]
                               //  [ 10 = Systemparameter schreibgeschützt ]
                               //    11 = Current hierachical block reference name
                               //  [ 12 = System parameter write-protected ]
                               //  [ 13 = System parameter write-protected ]
                               //    14 = Autosave path name
      string;                  // Parameter value
      );
```

Beschreibung

Die Funktion **scm_setstrpar** dient dazu, Systemparameter vom Typ `string` im **Schaltplaneditor** zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **scm_setstrpar** gesetzten Systemparametern können mit der Funktion **scm_getstrpar** abgefragt werden.

Siehe auch

Funktionen Funktionen **scm_getdblpar**, **scm_getintpar**, **scm_getstrpar**, **scm_setdblpar**, **scm_setintpar**.

scm_settagdata - Schaltplan Tagsymbolpin Zieldaten setzen (SCM)

Synopsis

```
int scm_settagdata(            // Status
      index C_FIGURE;          // Tag Element
      string;                  // Tag Pinname
      string;                  // Tag Referenzname 1
      string;                  // Tag Referenzname 2
      );
```

Beschreibung

Die Funktion **scm_settagdata** dient der Zuweisung der spezifizierten Tagzieldaten (Tagpinname und Tagreferenzbezeichnungen) an das angegebene Tagelement im Schaltplan. Der Funktionsrückgabewert ist Null bei erfolgreicher Zuweisung oder ungleich Null im Fehlerfall.

Siehe auch

Funktion **cap_gettagdata**.

scm_storecon - SCM Verbindung platzieren (SCM)

Synopsis

```
int scm_storecon(               // Status
      );
```

Beschreibung

Die Funktion **scm_storecon** erzeugt aus der internen Punktliste eine Verbindung auf dem aktuell geladenen Stromlaufblatt. Der Rückgabewert ist gleich Null, wenn die Verbindung erfolgreich platziert wurde, (-1) wenn kein gültiges Element geladen ist, (-2) wenn der Verbindungspunktmarker nicht definiert ist, oder (-3) bei ungültigen Polygondaten (nicht-orthogonale Segmente bzw. Kreisbögen in der Punktliste).

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in **forall**-Schleifen zur Iteration von **C_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

scm_storelabel - SCM Label platzieren (SCM)

Synopsis

```
int scm_storelabel(             // Status
      string;                   // Label Netzname
      int [0,2];                // Labeltyp:
                                //    0 = Standard Label
                                //    1 = Modulport
                                //    2 = Bustap
      double;                   // X-Koordinate (STD2)
      double;                   // Y-Koordinate (STD2)
      double;                   // Drehwinkel (STD3)
      int [0,1];                // Spiegelungsmodus (STD14)
      );
```

Beschreibung

Die Funktion **scm_storelabel** platziert einen Label mit den angegebenen Parametern auf dem aktuell geladenen Stromlaufblatt. Das erste Zeichen des Label-Netznamens darf kein Fragezeichen sein, da dieses zur Kennzeichnung von Modulports verwendet wird. Der Drehwinkel wird beim Platzieren von Bustaps ignoriert; außerdem müssen Bustaps immer an Bus-Verbindungssegmente angeschlossen, d.h. exakt auf solchen platziert werden. Der Name des Label-Bibliothekssymbols ergibt sich bei Standardlabels aus dem Netznamen bzw. zu **standard**, wenn kein entsprechendes Labelsymbol definiert ist; beim Platzieren von Modulports bzw. Bustaps werden per Default die Labelsymbole **port** bzw. **bustap** verwendet. Der Rückgabewert ist gleich Null, wenn der Label erfolgreich platziert wurde, (-1) wenn kein gültiges Element geladen ist, (-2) bei fehlenden/ungültigen Parametern, (-3) wenn der Label nicht ladbar ist, (-4) wenn die Labeldaten nicht in die Jobdatenbank kopiert werden konnten, oder (-5) wenn die Platzierungsdaten eines Bustaps nicht auf einem gültigen Verbindungssegment liegen.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in **forall**-Schleifen zur Iteration von **C_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

scm_storepart - SCM Bauteil platzieren (SCM)

Synopsis

```
int scm_storepart(             // Status
      & string;                // Bauteilname
      string;                  // Bauteil Bibliotheksteilname
      double;                  // X-Koordinate (STD2)
      double;                  // Y-Koordinate (STD2)
      double;                  // Drehwinkel (STD3)
      int [0,7];               // Spiegelungsmodus und Tagpintyp
                               //  (STD14|(CAP6<<1))
      );
```

Beschreibung

Die Funktion **scm_storepart** platziert ein Bauteil mit den angegebenen Parametern auf dem gegenwärtig geladenen SCM-Element. Wird eine Leerzeichenkette für den Bauteilnamen übergeben, so wird der Bauteilname automatisch erzeugt und im entsprechenden Parameter an den Aufrufer zurückgegeben. Der Rückgabewert ist gleich Null, wenn das Bauteil erfolgreich platziert wurde, (-1) wenn kein gültiges Element geladen ist, (-2) bei fehlenden/ungültigen Parametern, (-3) wenn das Bauteil nicht ladbar ist, (-4) wenn die Bauteildaten nicht in die Jobdatenbank kopiert werden konnten, (-5) wenn das Bauteil bereits platziert ist, oder (-6) wenn bei der automatischen Bauteilnamenserzeugung ein Fehler aufgetreten ist.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **C_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

scm_storepoly - SCM Fläche platzieren (SCM)

Synopsis

```
int scm_storepoly(             // Status
      int [0,5];               // Polygontyp (CAP2)
      );
```

Beschreibung

Die Funktion **scm_storepoly** generiert aus der mit **bae_storepoint** erzeugten internen Punktliste unter Verwendung der angegebenen Parameter ein Polygon auf dem gegenwärtig geladenen SCM-Element. Der Rückgabewert ist gleich Null, wenn das Polygon erfolgreich platziert wurde, (-1) wenn kein gültiges Element geladen ist, (-2) bei ungültigen Parametern, oder (-3) wenn die Punktliste für den gegebenen Polygontyp ungültig ist.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **C_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

scm_storetext - SCM Text platzieren (SCM)

Synopsis

```
int scm_storetext(             // Status
      string;                  // Textzeichenkette
      double;                  // Text-X-Koordinate (STD2)
      double;                  // Text-Y-Koordinate (STD2)
      double;                  // Textdrehwinkel (STD3)
      double ]0.0,[;           // Textgröße (STD2)
      int [0,1];               // Textspiegelungsmodus (STD14)
      int [0,[;                // Textmodus/-stil (CAP1|CAP7)
      );
```

Beschreibung

Die Funktion **scm_storetext** platziert einen Text mit den angegebenen Parametern auf dem gegenwärtig geladenen SCM-Element. Der Rückgabewert ist ungleich Null, wenn ungültige Daten übergeben wurden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **C_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden. Es können maximal 40 Zeichen der übergebenen Textzeichenkette gespeichert werden. Bei Übergabe längerer Zeichenketten gibt die Funktionen den Fehlerstatus zur Kennzeichnung ungültiger Parameter zurück.

Siehe auch

Funktion **scm_attachtextpos**.

C.4 PCB-Design-Systemfunktionen

In diesem Abschnitt werden (in alphabetischer Reihenfolge) die in der **Bartels User Language** definierten PCB-Design-Systemfunktionen beschrieben. Beachten Sie bitte die Konventionen zur Funktionsbeschreibung in Anhang C.1.

C.4.1 Layout-Datenzugriffsfunktionen

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp LAY zugeordnet, d.h. diese Funktionen können im **Layouteditor**, im **Autorouter** und im **CAM-Prozessor** aufgerufen werden:

lay_defelemname - Layout Setup default Elementname (LAY)

Synopsis

```
string lay_defelemname(       // Default Layout Elementname
      );
```

Beschreibung

Der Rückgabewert der Funktion **lay_defelemname** entspricht dem in der Setupdatei eingestellten Defaultnamen für Layoutplanelemente.

lay_deflibname - Layout Setup default Bibliothek (LAY)

Synopsis

```
string lay_deflibname(        // Default Layout Bibliotheksname
      );
```

Beschreibung

Der Rückgabewert der Funktion **lay_deflibname** entspricht dem in der Setupdatei eingestellten Defaultnamen für die Layoutbibliothek.

lay_defusrunit - Layout Setup default Benutzereinheitensystem (LAY)

Synopsis

```
int lay_defusrunit(           // Koordinatenanzeigemodus (STD7)
      );
```

Beschreibung

Der Rückgabewert der Funktion **lay_defusrunit** entspricht der in der Setupdatei angegebenen Defaulteinstellung für die Koordinatenanzeige (1=Inch, 0=mm).

lay_doclayindex - Layout Dokumentarlagenanzeigeindex (LAY)

Synopsis

```
int lay_doclayindex(          // Dokumentarlagen Anzeigeindex
      int [0,99];             // Dokumentarlagennummer
      );
```

Beschreibung

Die Funktion **lay_doclayindex** ermittelt den Dokumentarlagenanzeigeindex für die übergebene Dokumentarlage. Der Funktionsrückgabewert ist (-1) wenn eine eine ungütige Lagennummer spezifiziert wurde.

Siehe auch
Funktionen **lay_doclayname**, **lay_doclayside**, **lay_doclaytext**.

lay_doclayname - Layout Setup Name Dokumentarlage (LAY)

Synopsis

```
string lay_doclayname(        // Dokumentarlagenname
      int [0,99];             // Dokumentarlagennummer
      );
```

Beschreibung

Der Rückgabewert der Funktion **lay_doclayname** entspricht dem in der Setupdatei eingestellten Namen für die übergebene Dokumentarlage oder der Nullzeichenkette, wenn eine ungültige Lage angegeben wurde.

Siehe auch

Funktionen **lay_doclayindex**, **lay_doclayside**, **lay_doclaytext**.

lay_doclayside - Layout Setup Seitenmodus Dokumentarlage (LAY)

Synopsis

```
int lay_doclayside(           // Dokumentarlagen Seitenmodus:
                              //    (-2) = falsche Lagennummer
                              //    (-1) = wählbar
                              //    ( 0) = Seite 1
                              //    ( 1) = Seite 2
                              //    ( 2) = Beide Seiten
      int [0,99];             // Dokumentarlagennummer
      );
```

Beschreibung

Der Rückgabewert der Funktion **lay_doclayside** entspricht dem in der Setupdatei eingestellten Seitenmodus für die übergebene Dokumentarlage. (-1) bedeutet Seite 1, Seite 2 und beide Seiten sind wählbar. 0, 1 und 2 bedeuten, dass für die entsprechende Dokumentarlage Seite 1, Seite 2 bzw. beide Seiten als Defaultwert eingestellt ist. Ein Rückgabewert von (-2) bedeutet, dass eine falsche Lagennummer spezifiziert wurde.

Siehe auch

Funktionen **lay_doclayindex**, **lay_doclayname**, **lay_doclaytext**.

lay_doclaytext - Layout Setup Textmodus Dokumentarlage (LAY)

Synopsis

```
int lay_doclaytext(           // Dokumentarlagen Textmodus (LAY2)
      int [0,99];             // Dokumentarlagennummer
      );
```

Beschreibung

Der Rückgabewert der Funktion **lay_doclaytext** entspricht dem in der Setupdatei eingestellten Textmodus (LAY2) für die übergebene Dokumentarlage. Ein Rückgabewert von (-1) bedeutet, dass eine falsche Lagennummer spezifiziert wurde.

Siehe auch

Funktionen **lay_doclayindex**, **lay_doclayname**, **lay_doclayside**.

lay_figboxtest - Layout Elementüberschneidung Rechteck prüfen (LAY)

Synopsis

```
int lay_figboxtest(            // Status
      index L_FIGURE;          // Element
      double;                  // Rechteck linke Grenze (STD2)
      double;                  // Rechteck untere Grenze (STD2)
      double;                  // Rechteck rechte Grenze (STD2)
      double;                  // Rechteck obere Grenze (STD2)
      );
```

Beschreibung

Die Funktion **lay_figboxtest** prüft, ob das angegebene Element das angegebene Rechteck schneidet. Der Rückgabewert ist ungleich Null, wenn die Elementgrenzen das angegebene Rechteck schneiden.

lay_findconpart - Layout Bauteil in Netzliste suchen (LAY)

Synopsis

```
int lay_findconpart(           // Status
      string;                  // Bauteilname
      & index L_CPART;         // Rückgabe Netzlistenbauteil
      );
```

Beschreibung

Die Funktion **lay_findconpart** sucht den angegebenen Bauteilnamen in der Netzliste und gibt den Bauteileintrag gegebenenfalls in dem Bauteilrückgabeparameter zurück. Der Rückgabewert dieser Funktion ist ungleich Null, wenn das Bauteil nicht gefunden wurde.

Siehe auch

Funktionen **lay_findconpartpin**, **lay_findcontree**.

lay_findconpartpin - Layout Bauteilpin in Netzliste suchen (LAY)

Synopsis

```
int lay_findconpartpin(        // Status
      string;                  // Pinname
      index L_CPART;           // Netzlistenbauteil
      & index L_CPIN;          // Rückgabe Netzlistenbauteilpin
      );
```

Beschreibung

Die Funktion **lay_findconpartpin** sucht den Bauteilpin mit dem angegebenen Namen auf dem spezifizierten Netzlistenbauteil und gibt den Bauteilpineintrag gegebenenfalls in dem Bauteilpinrückgabeparameter zurück. Der Rückgabewert dieser Funktion ist ungleich Null, wenn der Bauteilpin nicht gefunden wurde.

Siehe auch
Funktionen **lay_findconpart**, **lay_findcontree**.

lay_findcontree - Layout Netz in Netzliste suchen (LAY)

Synopsis

```
int lay_findcontree(          // Status
      string;                 // Netzname
      & index L_CNET;         // Rückgabe Netzlisteneintrag
      );
```

Beschreibung

Die Funktion **lay_findcontree** sucht den angegebenen Netznamen in der Netzliste und gibt den Netzlisteneintrag ggf. in dem Netzrückgabeparameter zurück. Der Rückgabewert dieser Funktion ist ungleich Null, wenn das Netz nicht gefunden wurde.

Siehe auch

Funktionen **lay_findconpart**, **lay_findconpartpin**.

lay_getplanchkparam - Layout DRC Abstände abfragen (LAY)

Synopsis

```
void lay_getplanchkparam(
      & double;               // Abstand Bahn zu Bahn (STD2)
      & double;               // Abstand Bahn zu Kupfer (STD2)
      & double;               // Abstand Kupfer zu Kupfer (STD2)
      & double;               // Standard-Leiterbahnbreite (STD2)
      & string;               // Blockname
      int [-6,99];            // Signallagencode (LAY1)
                              //  (Lage!=(-1) nur in BAE HighEnd zulässig)
      int [0,0[;              // DRC-Blocknummer
      );
```

Beschreibung

Die Funktion **lay_getplanchkparam** gibt in den Parametern die Abstandsparameter für den Design Rule Check (DRC) im **Layouteditor** zurück. In **BAE Professional**, **BAE Economy** und **BAE Light** können nur die globalen Parameter für den Lagencode -1 (Alle Lagen) und den DRC-Block 0 abgefragt werden. In **BAE HighEnd** können darüber hinaus auch Parameterblöcke ausgelesen werden, die lagenspezifischer Mindestabstände für beliebige Signallagen (Lagencodes 0 bis 99), die oberste Lage (Lagencode -5) sowie die Innenlagen (Lagencode -6) definieren. Der Parameterblock 0 ist immer vorhanden und enthält die global definierten DRC-Parameter. Ebenso ist auch immer für den Lagencode -1 (Alle Lagen) eine Vorgabe vorhanden.

Siehe auch

Funktion **lay_setplanchkparam**.

lay_getpowplanetree - Layout Netznummer in Versorgungslage abfragen (LAY)

Synopsis

```
int lay_getpowplanetree(      // Netznummer oder (-1)
      double;                 // X-Koordinate (STD2)
      double;                 // Y-Koordinate (STD2)
      int;                    // Versorgunglagennummer (LAY1)
      double;                 // Bohrdurchmesser (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **lay_getpowplanetree** entspricht der Netznummer des über die angegebene Versorgungslagenkoordinate angeschlossenen Signalpegels (**L_CNET**.NUMBER). Die Auswertung erfolgt auf der angegebenen Versorgungslage unter berücksichtigung des spezifizierten Bohrdurchmessers. Diese Funktion wird zur Ermittlung von Netzanschlüssen bei der Verwendung geteilter Potentiallagen (Split Power Planes) benötigt. Der Funktionsrückgabewert ist (-1), wenn kein Signalanschluss gefunden wurde bzw. wenn eine ungültige Versorgungslagennummer spezifiziert wurde.

Warnung

Die an **lay_getpowplanetree** übergebenen Koordinatenangaben werden als Absolutkoordinaten auf dem aktuell geladenen Element interpretiert. Beim indirekten Aufruf von **lay_getpowplanetree** über **lay_scan***-Funktionen muss deshalb der spezifizierte Scan-Offset entsprechend berücksichtigt werden.

lay_getpowpolystat - Layout Versorgungslagenpolygonstatus abfragen (LAY)

Synopsis

```
int lay_getpowpolystat(       // Status
      index L_FIGURE;         // Figurenlistenelement - Versorgungslagenpolygon
      & int;                  // Rückgabeflag:
                              //    Polgyon schneidet Platinenumrandung
      & int;                  // Rückgabeflag:
                              //    Polygon schneidet andere
Versorgungslagenpolygone
      );
```

Beschreibung

Mit der Funktion **lay_getpowpolystat** wird der Status des angegebenen Versorgungslagenpolygons (Split Power Plane) abgefragt. Die Rückgabeflags geben an, ob das Versorgungslagenpolygon die Platinenumrandung oder andere Versorgungslagenpolygone schneidet. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage, (-1) bei ungültigen Parametern oder ungültiger Interpreterumgebung oder (-2) wenn das angegebene Figurenlistenelement kein Versorgungslagenpolygon ist.

lay_getrulecnt - Layoutelement Regelanzahl abfragen (LAY)

Synopsis

```
int lay_getrulecnt(           // Regelanzahl oder (-1) bei Fehler
      int;                    // Object class code
      int;                    // Object ident code (int oder Indextyp)
      );
```

Beschreibung

Mit der Funktion **lay_getrulecnt** kann die Anzahl der an ein spezifisches Objekt zugewiesenen Regeln ermittelt werden. Die Abfrage kann für das aktuell geladene Element (Objektklasse 0 mit **int**-Wert 0 für die Objektidentifikation), ein Element aus der Figurenliste des aktuell geladenen Elements (Objektklasse 1 mit gültigem Wert vom Typ index **L_FIGURE** für die Objektidentifikation) oder ein Poolelement (Objektklasse 2 mit gültigem Wert vom Typ index **L_POOL** für die Objektidentifikation) durchgeführt werden. Die von **lay_getrulecnt** ermittelte (nicht-negative) objektspezifische Regelanzahl wird im Rückgabewert der Funktion übergeben und bestimmt den Wertebereich für den Regelnamenslistenindex in nachfolgenden Aufrufen der Funktion **lay_getrulename** zur Ermittlung von Regelnamen für das entsprechende Objekt. Der Rückgabewert ergibt sich zu (-1), wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **lay_ruleerr** ermittelt werden.

Siehe auch

Funktionen **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplanatt**, **lay_ruleplandet**, **lay_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

lay_getrulename - Layoutelement Regelname abfragen (LAY)

Synopsis

```
int lay_getrulename(            // Status
      int;                      // Object class code
      int;                      // Object ident code (int oder Indextyp)
      int [0,[;                 // Regelnamenslistenindex
      & string;                 // Regelname
      );
```

Beschreibung

Mit der Funktion **lay_getrulename** können die Namen der an ein spezifisches Objekt zugewiesenen Regeln ermittelt werden. Die Abfrage kann für das aktuell geladene Element (Objektklasse 0 mit `int`-Wert 0 für die Objektidentifikation), ein Element aus der Figurenliste des aktuell geladenen Elements (Objektklasse 1 mit gültigem Wert vom Typ index **L_FIGURE** für die Objektidentifikation) oder ein Poolelement (Objektklasse 2 mit gültigem Wert vom Typ index **L_POOL** für die Objektidentifikation) durchgeführt werden. Der Regelnamenslistenindex zur Auswahl der gewünschten Regel muss mindestens Null jedoch kleiner als die mit der Funktion **lay_getrulecnt** abfragbare Anzahl objektspezifischer Regeln sein. Der ermittelte Regelname wird über den letzten Funktionsparameter an den Aufrufer zurückgegeben. Der Rückgabewert der Funktion **lay_getrulename** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **lay_ruleerr** ermittelt werden.

Siehe auch

Funktionen **lay_getrulecnt**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplanatt**, **lay_ruleplandet**, **lay_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

lay_getscclass - Aktuell gescannte Layoutelementklasse ermitteln (LAY)

Synopsis

```
int lay_getscclass(             // Rückgabe Layoutelementklasse:
                                //    0 = Layout
                                //    1 = Bauteil
                                //    2 = Padstack
                                //    3 = Pad
                                //    (-1) sonst
      );
```

Beschreibung

Die Funktion **lay_getscclass** gibt die aktuell gescannte Layoutelementklasse zurück. Der Aufruf dieser Funktion ist nur innerhalb der Callbackfunktionen von **lay_scanall**, **lay_scanfelem** oder **lay_scanpool** sinnvoll. An anderer Stelle, oder wenn gerade kein Layoutelement gescannt wird, ergibt sich der Funktionsrückgabewert zu (-1).

Siehe auch

Funktionen **lay_scanall**, **lay_scanfelem**, **lay_scanpool**.

lay_getscpartrpidx - Aktuell gescanntes benannte Layoutbauteil ermitteln (LAY)

Synopsis

```
index L_NREF lay_getscpartrpidx   // Rückgabe Bauteilindex oder (-1)
      );
```

Beschreibung

Die Funktion **lay_getscpartrpidx** gibt das aktuell gescannte Layoutbauteil als benannte Referenz zurück, d.h. damit kann beim Scannen von Leiterbahnen, Polygonen, Texten, Pins, etc. die Bauteilzugehörigkeit des jeweils gescannten Objects ermittelt werden. Der Aufruf dieser Funktion ist nur innerhalb der Callbackfunktionen von **lay_scanall**, **lay_scanfelem** oder **lay_scanpool** sinnvoll. An anderer Stelle, oder wenn gerade kein Bauteil gescannt wird, ergibt sich der Funktionsrückgabewert zu (-1).

Siehe auch

Funktionen **lay_scanall**, **lay_scanfelem**, **lay_scanpool**.

lay_getscrefpidx - Aktuell gescanntes Layoutbibliothekselement ermitteln (LAY)

Synopsis

```
index L_POOL lay_getscrefpidx     // Rückgabe Poolindex, oder (-1) wenn kein Makro
      );
```

Beschreibung

Die Funktion **lay_getscrefpidx** gibt den Poolindex des aktuell gescannten Bibliothekselements zuück, d.h. damit kann beim Scannen von Leiterbahnen, Polygonen, Texten, etc. die Zugehörigkeit des jeweils gescannten Objects zum entsprechenden Bibliothekselement (Bauteil, Padstack, Pad) ermittelt werden. Der Aufruf dieser Funktion ist nur innerhalb der Callbackfunktionen von **lay_scanall**, **lay_scanfelem** oder **lay_scanpool** sinnvoll. An anderer Stelle, oder wenn gerade kein Bibliothekselement gescannt wird, ergibt sich der Funktionsrückgabewert zu (-1).

Siehe auch

Funktionen **lay_scanall**, **lay_scanfelem**, **lay_scanpool**.

lay_getscstkcnt - Layout Scanfunktion Stacktiefe abfragen (LAY)

Synopsis

```
int lay_getscstkcnt(           // Scan-Stacktiefe
      );
```

Beschreibung

Die Funktion **lay_getscstkcnt** dient der Abfrage der aktuellen Stacktiefe der Layoutscanfunktionen. **lay_getscstkcnt** kann somit zur Abarbeitungskontrolle innerhalb der Callbackfunktionen von **lay_scanall**, **lay_scanfelem** oder **lay_scanpool** eingesetzt werden.

Siehe auch

Funktionen **lay_scanall**, **lay_scanfelem**, **lay_scanpool**.

lay_getsctextdest - Zielpunkt des gescannten Layouttextes abfragen (LAY)

Synopsis

```
int lay_getsctextdest(         // Status
      & double;                // Rückgabe Textlinienzielpunkt X-Koordinate (STD2)
      & double;                // Rückgabe Textlinienzielpunkt Y-Koordinate (STD2)
      );
```

Beschreibung

Mit der Funktion **lay_getsctextdest** kann der Ziel- bzw. Endpunkt der Textbasislinie des aktuell gescannten Textelements ermittelt werden. Der Aufruf dieser Funktion ist nur innerhalb der Callbackfunktionen von **lay_scanall**, **lay_scanfelem** oder **lay_scanpool** sinnvoll. Der Funktionsrückgabewert ist 1 bei erfolgreicher Abfrage oder Null bei fehlgschlagener Abfrage.

Siehe auch

Funktionen **lay_scanall**, **lay_scanfelem**, **lay_scanpool**.

lay_gettreeidx - Layout Netznummer in Netzliste suchen (LAY)

Synopsis

```
int lay_gettreeidx(            // Status
      int;                     // Netznummer
      & index L_CNET;          // Rückgabe Netzlisteneintrag
      );
```

Beschreibung

Die Funktion **lay_gettreeidx** sucht die angegebene Netznummer in der Netzliste und gibt den Netzlisteneintrag gegebenenfalls in dem Netzrückgabeparameter zurück. Der Rückgabewert ist ungleich Null, wenn die Netznummer nicht gefunden wurde.

lay_grpdisplay - Layout Setup Gruppenlage abfragen (LAY)

Synopsis

```
int lay_grpdisplay(             // Dokumentarlagennummer (LAY1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **lay_grpdisplay** entspricht der in der Setupdatei eingestellten Dokumentarlage für die Gruppendarstellung.

lay_lastfigelem - Zuletzt modifiziertes Layoutelement ermitteln (LAY)

Synopsis

```
int lay_lastfigelem(            // Status
      & index L_FIGURE;         // Rückgabe Element
      );
```

Beschreibung

Die Funktion **lay_lastfigelem** ermittelt das zuletzt erzeugte bzw. modifizierte Layoutelement und übergibt den entsprechenden Index aus der Figurenliste im Rückgabeparameter. Der Rückgabewert der Funktion ist Null wenn ein derartiges Element existiert, oder ungleich Null andernfalls.

lay_maccoords - Layout Makrokoordinaten abfragen (LAY)

Synopsis

```
void lay_maccoords(
      & double;                 // X-Position (STD2)
      & double;                 // Y-Position (STD2)
      & double;                 // Drehwinkel (STD3)
      & int;                    // Spiegelungsmodus (STD14)
      & int;                    // Lage (LAY1 für Pad auf Padstack)
      );
```

Beschreibung

Die Funktion **lay_maccoords** gibt in den Parametern die Platzierungsdaten für das aktuell bearbeitete Makro zurück. Der Aufruf dieser Funktion ist nur innerhalb der Makroscanfunktion von **lay_scanall**, **lay_scanfelem** oder **lay_scanpool** sinnvoll. An anderer Stelle werden Null-Defaultwerte zurückgegeben.

Siehe auch

Funktionen **lay_scanall**, **lay_scanfelem**, **lay_scanpool**.

lay_macload - Layoutsymbol in den Arbeitsspeicher laden (LAY)

Synopsis

```
int lay_macload(                // Status
      & index L_POOL;           // Makro Poolelementindex
      string;                   // DDB-Dateiname
      string;                   // Elementname
      int [100,[;               // Element DDB-/Datenbankklasse (STD1)
      );
```

Beschreibung

Die Funktion **lay_macload** lädt das angegebene Layoutsymbol in den Arbeitsspeicher und übergibt den zugehörigen Poolindex im entsprechenden Parameter. Der Funktionsrückgabewert ist Null, wenn das Symbol erfolgreich geladen wurde, (-1) bei Dateizugriffsfehlern, (-2) bei fehlenden oder ungültigen Parameterangaben oder 1 wenn referenzierte Bibliothekselemente nicht verfügbar sind. **lay_macload** ist für die Anwendung in Funktionen zur Auswertung von Bibliotheksdateien konzipiert. Mit der Funktion **lay_macrelease** können Layoutsymbole wieder aus dem Arbeitsspeicher entfernt werden.

Siehe auch

Funktion **lay_macrelease**.

lay_macrelease - Layoutsymbol aus dem Arbeitsspeicher löschen (LAY)

Synopsis

```
void lay_macrelease(
      index L_POOL;           // Makro Poolelementindex
      );
```

Beschreibung

Die Funktion **lay_macrelease** löscht das über den Poolelementindex spezifizierte Layoutsymbol aus dem Arbeitsspeicher. **lay_macrelease** ist als Pendant zur Funktion **lay_macload** konzipiert.

Siehe auch

Funktion **lay_macload**.

lay_menulaylinecnt - Lagenmenüzeilenanzahl abfragen (LAY)

Synopsis

```
int lay_menulaylinecnt(       // Anzahl Einträge im Signallagenmenü
      );
```

Beschreibung

Der Rückgabewert der Funktion **lay_menulaylinecnt** entspricht der aktuell definierten Anzahl der Menüzeilen im Signallagenmenü des Layoutsystems. Das Signallagenmenü kann mit Hilfe des Utilityprogramms **BSETUP** konfiguriert werden, wobei bis zu 12 Signallagen mit Lagennummer und Lagenname definiert werden können.

Siehe auch

Funktionen **lay_menulaylinelay**, **lay_menulaylinename**.

lay_menulaylinelay - Lagennummer der angegebenen Lagenmenüzeile abfragen (LAY)

Synopsis

```
int lay_menulaylinelay(       // Menüzeile Lagennummer
      int [0,11];             // Menüzeilennummer
      );
```

Beschreibung

Der Rückgabewert der Funktion **lay_menulaylinelay** entspricht der Lagennummer, die für die spezifizierte Menüzeile im Signallagenmenü des Layoutsystems aktuell definiert ist. Das Signallagenmenü kann mit Hilfe des Utilityprogramms **BSETUP** konfiguriert werden, wobei bis zu 12 Signallagen mit Lagennummer und Lagenname definiert werden können.

Siehe auch

Funktionen **lay_menulaylinecnt**, **lay_menulaylinename**.

lay_menulaylinename - Lagenname der angegebenen Lagenmenüzeile abfragen (LAY)

Synopsis

```
string lay_menulaylinename(   // Menüzeile Lagenbezeichnung
      int [0,11];             // Menüzeilennummer
      );
```

Beschreibung

Der Rückgabewert der Funktion **lay_menulaylinename** entspricht dem Lagennamen, der für die spezifizierte Menüzeile im Signallagenmenü des Layoutsystems aktuell definiert ist. Das Signallagenmenü kann mit Hilfe des Utilityprogramms **BSETUP** konfiguriert werden, wobei bis zu 12 Signallagen mit Lagennummer und Lagenname definiert werden können.

Siehe auch

Funktionen **lay_menulaylinecnt**, **lay_menulaylinelay**.

lay_nrefsearch - Layout Name auf Plan suchen (LAY)

Synopsis

```
int lay_nrefsearch(            // Status
      string;                  // Bauteilname
      & index L_FIGURE;        // Rückgabe Element
      );
```

Beschreibung

Die Funktion **lay_nrefsearch** prüft, ob das angegebene Bauteil platziert ist und gibt gegebenenfalls das zugehörige Element zurück. Der Rückgabewert ist ungleich Null, wenn das Bauteil nicht gefunden wurde.

lay_planmidlaycnt - Layout Innenlagenanzahl abfragen (LAY)

Synopsis

```
int lay_planmidlaycnt(         // Innenlagenanzahl
      );
```

Beschreibung

Die Funktion **lay_planmidlaycnt** ermittelt die Anzahl der Innenlagen für des aktuell geladenen Layouts.

Siehe auch

Funktion **lay_plantoplay**.

lay_plantoplay - Layout oberste Lage abfragen (LAY)

Synopsis

```
int lay_plantoplay(            // Oberste Lage (LAY1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **lay_plantoplay** entspricht der im aktuell geladenen Element gültigen Einstellung der obersten Lage (LAY1), bzw. Signallage 2, wenn kein Layoutplan geladen ist.

Siehe auch

Funktion **lay_planmidlaycnt**.

lay_pltmarklay - Layout Setup Passermarkenlage abfragen (LAY)

Synopsis

```
int lay_pltmarklay(            // Dokumentarlagennummer
      );
```

Beschreibung
Der Rückgabewert der Funktion **lay_pltmarklay** entspricht der in der Setupdatei eingestellten Passermarkenlage.

lay_ruleerr - Layout-Regelsystem Fehlerstatus abfragen (LAY)

Synopsis

```
void lay_ruleerr(
      & int;              // Fehlercode
      & string;           // Fehlerstring
      );
```

Beschreibung

Die Funktion **lay_ruleerr** dient der Ermittlung des Regelsystemstatus, d.h. die Funktion **lay_ruleerr** kann zur genauen Bestimmung der Fehlerursache im Falle eines fehlerhaften Aufrufs einer Regelsystemfunktion verwendet werden.

Diagnose

Zur Bestimmung der Fehlerursache sind die durch **lay_ruleerr** zurückgegebenen Parameterwerte heranzuziehen. Der zurückgegebene Fehlerstring dient ggf. der Identifizierung des fehlerverursachenden Elements. Die möglichen Werte, die der Fehlercode durch die Ausführung eines Regelsystemfunktion annehmen kann, haben folgende Bedeutung:

Fehlercode	**Bedeutung**
0	Regelsystem Operation/Funktion erfolgreich beendet
1	Regelsystem Hauptspeicher nicht ausreichend
2	Regelsystem Interner Fehler **<e>**
3	Regelsystem Funktionsparameter ungültig
128	Regelsystem Datenbankdatei kann nicht angelegt werden
129	Regelsystem Datenbankdatei Lese-/Schreibfehler
130	Regelsystem Datenbankdatei von falschem Typ
131	Regelsystem Datenbankdateistruktur beschädigt
132	Regelsystem Datenbankdatei nicht gefunden
133	Regelsystem Datenbankfehler allgemein (Interner Fehler)
134	Regelsystem Regel **<r>** nicht Regeldatenbank gefunden
135	Regelsystem Regel in falschem Format in Datenbank (Interner Fehler **<e>**)
136	Regelsystem Objekt nicht gefunden
137	Regelsystem Objekt mehrfach definiert (Interner Fehler)
138	Regelsystem Inkompatible Definition der Variable **<v>**
139	Regelsystem Regel **<r>** mit inkompatibler Compiler-Version übersetzt

Der Fehlerstring kann je nach Fehlerfall eine Regel **<r>**, eine Variable **<v>** oder einen (internen) Fehlerstatus **<e>** bezeichnen. Datenbankdateifehler beziehen sich auf Probleme beim Zugriff auf die Regeldatenbankdatei **brules.vdb** im BAE-Programmverzeichnis. Interne Fehler weisen üblicherweise auf Implementierungslücken im Regelsystem hin und sollten in jedem Fall an Bartels gemeldet werden.

Siehe auch

Funktionen **lay_getrulecnt**, **lay_getrulename**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplanatt**, **lay_ruleplandet**, **lay_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

lay_rulefigatt - Regelzuweisung an Layout-Figurenelement (LAY)

Synopsis

```
int lay_rulefigatt(            // Status
      index L_FIGURE;          // Figurenlistenelement
      void;                    // Regelname oder Regelnamensliste
      );
```

Beschreibung

Die Funktion **lay_rulefigatt** erlaubt die Zuweisung von Regeln an das mit dem ersten Funktionsparameter spezifizierte Figurenlistenelement des aktuell geladenes Elements. Der zweite Funktionsparameter erlaubt dabei sowohl die Spezifikation eines einzelnen Regelnamens (d.h. eines Wertes vom Typ `string`) als auch die Angabe einer ganzen Liste von Regelnamen (d.h. eines Arrays vom Typ `string`). Beachten Sie, dass vor der Zuweisung des angegebenen Regelsatzes zunächst alle bestehenden Regelzuweisungen an das spezifizierte Figurenlistenelement gelöscht werden. Der Rückgabewert der Funktion **lay_rulefigatt** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **lay_ruleerr** ermittelt werden.

Siehe auch

Funktionen **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplanatt**, **lay_ruleplandet**, **lay_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

lay_rulefigdet - Regelzuweisungen von Layout-Figurenelement lösen (LAY)

Synopsis

```
int lay_rulefigdet(            // Status
      index L_FIGURE;          // Figurenlistenelement
      );
```

Beschreibung

Die Funktion **lay_rulefigdet** löscht *alle* aktuell bestehenden Regelzuweisungen an das über den Funktionsparameter spezifizierte Figurenlistenelement des aktuell geladenen Elements. Der Rückgabewert der Funktion **lay_rulefigdet** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **lay_ruleerr** ermittelt werden.

Siehe auch

Funktionen **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplanatt**. **lay_ruleplandet**, **lay_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

lay_rulelaysatt - Regelzuweisung an Layoutlagenaufbau (LAY)

Synopsis

```
int lay_rulelaysatt(           // Status
      int [0,111];             // Lagenaufbauindex
      void;                    // Regelname oder Regelnamensliste
      );
```

Beschreibung

Die Funktion **lay_rulelaysatt** erlaubt die Zuweisung von Regeln an den durch den angegebenen Lagenaufbauindex spezifizierten Lagenaufbau. Der Funktionsparameter zur Regelangabe erlaubt dabei sowohl die Spezifikation eines einzelnen Regelnamens (d.h. eines Wertes vom Typ `string`) als auch die Angabe einer ganzen Liste von Regelnamen (d.h. eines Arrays vom Typ `string`). Beachten Sie, dass vor der Zuweisung des angegebenen Regelsatzes zunächst alle bestehenden Regelzuweisungen des bearbeiteten Lagenaufbaus gelöscht werden. Der Rückgabewert der Funktion **lay_rulelaysatt** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **lay_ruleerr** ermittelt werden.

Siehe auch

Funktionen **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysdet**, **lay_ruleplanatt**, **lay_ruleplandet**, **lay_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

lay_rulelaysdet - Regelzuweisungen von Layoutlagenaufbau lösen (LAY)

Synopsis

```
int lay_rulelaysdet(          // Status
      int [0,111];            // Lagenaufbauindex
      );
```

Beschreibung

Die Funktion **lay_rulelaysdet** löscht *alle* aktuell bestehenden Regelzuweisungen an den durch den Lagenaufbauindex angegebenen Lagenaufbau. Der Rückgabewert der Funktion **lay_rulelaysdet** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **lay_ruleerr** ermittelt werden.

Siehe auch

Funktionen **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_ruleplanatt**, **lay_ruleplandet**, **lay_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

lay_ruleplanatt - Regelzuweisung an aktuell geladenes Layoutelement (LAY)

Synopsis

```
int lay_ruleplanatt(          // Status
      void;                   // Regelname oder Regelnamensliste
      );
```

Beschreibung

Die Funktion **lay_ruleplanatt** erlaubt die Zuweisung von Regeln an das aktuell geladenes Elements. Der Funktionsparameter erlaubt dabei sowohl die Spezifikation eines einzelnen Regelnamens (d.h. eines Wertes vom Typ `string`) als auch die Angabe einer ganzen Liste von Regelnamen (d.h. eines Arrays vom Typ `string`). Beachten Sie, dass vor der Zuweisung des angegebenen Regelsatzes zunächst alle bestehenden Regelzuweisungen an das aktuelle Element gelöscht werden. Der Rückgabewert der Funktion **lay_ruleplanatt** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **lay_ruleerr** ermittelt werden.

Siehe auch

Funktionen **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplandet**, **lay_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

lay_ruleplandet - Regelzuweisungen von aktuell geladenem Layoutelement lösen (LAY)

Synopsis

```
int lay_ruleplandet(          // Status
      );
```

Beschreibung

Die Funktion **lay_ruleplandet** löscht *alle* aktuell bestehenden Regelzuweisungen an das aktuell geladene Element. Der Rückgabewert der Funktion **lay_ruleplandet** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **lay_ruleerr** ermittelt werden.

Siehe auch

Funktionen **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplanatt**, **lay_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

lay_rulequery - Layoutelement Regelabfrage durchführen (LAY)

Synopsis

```
int lay_rulequery(              // Trefferanzahl oder (-1) bei Fehler
      int;                      // Object class code
      int;                      // Object ident code (int oder Indextyp)
      string;                   // Subjektname
      string;                   // Prädikatname
      string;                   // Abfragekommando
      & void;                   // Abfrageergebnis
      []                        // Optionale Abfrageparameter
      );
```

Beschreibung

Die Funktion **lay_rulequery** führt eine Regelabfrage für ein spezifisches Objekt durch. Die Abfrage kann für das aktuell geladene Element (Objektklasse 0 mit **int**-Wert 0 für die Objektidentifikation), ein Element aus der Figurenliste des aktuell geladenen Elements (Objektklasse 1 mit gültigem Wert vom Typ index **L_FIGURE** für die Objektidentifikation) oder ein Poolelement (Objektklasse 2 mit gültigem Wert vom Typ index **L_POOL** für die Objektidentifikation) durchgeführt werden. Zur Durchführung der Abfrage müssen sowohl ein Regelsubjekt als auch ein Regelprädikat namentlich angegeben werden. Zusätzlich ist ein Abfragekommando zu spezifizieren. Das Abfragekommando kann Platzhalter für Wertvorgaben und einen Abfrageoperator enthalten. Folgende Abfrageoperatoren stehen zur Verfügung:

?d	zur Abfrage von **int**-Werten
?f	zur Abfrage von **double**-Werten
?s	zur Abfrage von **string**-Werten

Dem Abfrageoperator kann wahlweise einer der folgenden Selektionsoperatoren vorangestellt werden:

+	zur Abfrage des Maximums aller gefundenen Werte
-	zur Abfrage des Minimums aller gefundenen Werte

Standardmäßig, d.h. bei Auslassung des Selektionsoperators wird der **+**-Operator verwendet. Der über die Abfrage gefundene Werteintrag wird im Funktionsparameter für das Abfrageergebnis zurückgegeben. Hierbei ist sicherzustellen, dass der Datentyp des Parameters für das Abfrageergebnis mit dem Abfragedatentyp übereinstimmt (**int** für **?d**, **double** für **?f**, **string** für **?s**). Neben dem Abfrageoperator können folgende Platzhalter für Wertvorgaben im Abfragekommando spezifiziert werden:

%d	zur Angabe von **int**-Werten
%f	zur Angabe von **double**-Werten
%s	zur Angabe von **string**-Werten

Für jeden im Abfragekommando spezifizierten Platzhalter für Wertvorgaben ist ein optionaler Abfrageparameter an die Funktion **lay_rulequery** zu übergeben. Die Reihenfolge dieser optionalen Parameter sowie deren Datentypen müssen mit den Spezifikationen im Abfragekommando übereinstimmen. Nach erfolgreicher Abarbeitung der Regelabfrage wird im Rückgabewert die (nicht-negative) Anzahl der gefundenen Einträge an den Aufrufer zurückgegeben. Der Rückgabewert ergibt sich zu (-1), wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **lay_ruleerr** ermittelt werden.

Beispiele

Sofern die Regel

```
rule somerule
{
    subject subj
    {
        pred := ("A", 2);
        pred := ("A", 4);
        pred := ("B", 1);
        pred := ("C", 3);
        pred := ("B", 6);
        pred := ("D", 5);
        pred := ("D", 6);
        pred := ("A", 3);
    }
}
```

definiert und dem aktuell geladenen Element zugewiesen ist, würde der **lay_rulequery**-Aufruf

```
hitcount = lay_rulequery(0,0,"subj","pred","%s ?d",intresult,"A") ;
```

die **`int`**-Variable **`hitcount`** auf 3 und die **`int`**-Variable **`intresult`** auf 4 setzen, während der Aufruf

```
hitcount = lay_rulequery(0,0,"subj","pred","-?s %d",strresult,6) ;
```

die Variable **`hitcount`** auf 2 und die **`string`**-Variable **`strresult`** auf **`B`** setzt.

Siehe auch

Funktionen **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplanatt**, **lay_ruleplandet**; **Neuronales Regelsystem** und Rule System Compiler.

lay_scanall - Layout Scan über alle Elemente (LAY)

Synopsis

```
int lay_scanall(                // Scan Status
      double;                   // X-Offset (STD2)
      double;                   // Y-Offset (STD2)
      double;                   // Drehwinkel (STD3)
      int [0,1];                // Element in Arbeitsbereich Flag (STD10)
      int [0,1];                // Connectivity Scan Flag:
                                //    0 = kein Scan
                                //    1 = Scan erlaubt
      * int;                    // Makrofunktion
      * int;                    // Polygonfunktion
      * int;                    // Leiterbahnfunktion
      * int;                    // Textfunktion
      * int;                    // Bohrungsfunktion
      * int;                    // Lagencheckfunktion
      * int;                    // Levelcheckfunktion
      );
```

Beschreibung

Die Funktion **lay_scanall** scannt alle auf dem aktuell geladenen Element platzierten Elemente über alle Hierarchiestufen. Dabei werden für alle gefundenen Elemente die referenzierten Anwenderfunktionen aufgerufen. Soll ein Funktionstyp nicht aufgerufen werden, so ist für den entsprechenden Parameter `NULL` anzugeben. Der Rückgabewert der Funktion **lay_scanall** ist ungleich Null, wenn ungültige Parameter angegeben wurden, oder wenn eine der referenzierten Anwenderfunktionen einen Fehler an die Funktion **lay_scanall** zurückgemeldet hat.

Makrofunktion

```
int macrofuncname(
      index L_MACRO macro,      // Makro Index
      index L_POOL pool,        // Pool Element Index
      int macinws,              // Makro in Arbeitsbereich Flag (STD10)
      string refname,           // Makro Referenzname
      index L_LEVEL level       // Makro Level
      )
{
      // Verarbeitungsprogramm
      :
      return(contscan);
}
```

Die Makroplatzierungsdaten können mit der Funktion **lay_maccoords** abgefragt werden. Der Rückgabewert dieser Funktion sollte gleich Null sein, wenn der Scan für dieses Makro nicht weitergeführt werden soll, (-1) wenn ein Fehler aufgetreten ist und 1 wenn weitergescannt werden soll.

Polygonfunktion

```
int polyfuncname(
      index L_POLY poly,        // Polygondaten
      int layer,                // Lage (LAY1)
      int polyinws,             // Polygon in Arbeitsbereich Flag (STD10)
      int tree,                 // Netznummer oder (-1)
      index L_LEVEL level       // Polygon Level
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Der Rückgabewert dieser Funktion sollte ungleich Null sein, wenn ein Fehler aufgetreten ist.

Leiterbahnfunktion

```
int pathfuncname(
      index L_LINE path,       // Leiterbahndaten
      int layer,               // Lage (LAY1)
      int pathinws,            // Bahn in Arbeitsbereich Flag (STD10)
      index L_LEVEL level      // Bahn Level
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Der Rückgabewert dieser Funktion sollte ungleich Null sein, wenn ein Fehler aufgetreten ist.

Textfunktion

```
int textfuncname(
      index L_TEXT text,       // Textdaten
      double x,                // X-Koordinate (STD2)
      double y,                // Y-Koordinate (STD2)
      double angle,            // Drehwinkel (STD3)
      int mirr,                // Spiegelung (STD14)
      int layer,               // Lage (LAY1)
      double size,             // Text Größe (STD2)
      string textst,           // Textzeichenkette
      int textinws             // Text in Arbeitsbereich Flag (STD10)
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Der Rückgabewert dieser Funktion sollte ungleich Null sein, wenn ein Fehler aufgetreten ist.

Bohrungsfunktion

```
int drillfuncname(
      index L_DRILL drill,     // Bohrungsdaten
      double x,                // Transformierte X-Koordinate (STD2)
      double y,                // Transformierte Y-Koordinate (STD2)
      int drillinws,           // Bohrung in Arbeitsbereich Flag (STD10)
      int tree,                // Netznummer oder (-1)
      index L_LEVEL level      // Bohrung Level
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Der Rückgabewert dieser Funktion sollte ungleich Null sein, wenn ein Fehler aufgetreten ist.

Lagencheckfunktion

```
int laycheckfuncname(
      int layer              // Lage (LAY1)
      )
{
      // Verarbeitungsprogramm
      :
      return(contscan);
}
```

Der Rückgabewert dieser Funktion sollte gleich Null sein, wenn der Scan bei der übergebenen Lage nicht weitergeführt werden soll, (-1) wenn ein Fehler aufgetreten ist und 1 wenn weitergescannt werden soll. Bei Beschränkung auf die interessierenden Lagen durch diese Funktion wird der Scanvorgang erheblich beschleunigt.

Levelcheckfunktion

```
int levcheckfuncname(
      index L_LEVEL level    // Level
      )
{
      // Verarbeitungsprogramm
      :
      return(contscan);
}
```

Der Rückgabewert dieser Funktion sollte gleich Null sein, wenn der Scan bei dem übergebenen Level nicht weitergeführt werden soll, (-1) wenn ein Fehler aufgetreten ist und 1 wenn weitergescannt werden soll. Bei Beschränkung auf die interessierenden Levels durch diese Funktion wird der Scanvorgang erheblich beschleunigt.

Siehe auch

Funktionen **lay_maccoords**, **lay_scanfelem**, **lay_scanpool**.

lay_scanfelem - Layout Scan über Figurenelement (LAY)

Synopsis

```
int lay_scanfelem(           // Scan Status
      index L_FIGURE;        // Figurenelement
      double;                // X-Offset (STD2)
      double;                // Y-Offset (STD2)
      double;                // Drehwinkel (STD3)
      int [0,1];             // Element in Arbeitsbereich Flag (STD10)
      int [0,1];             // Connectivity Scan Flag:
                             //    0 = kein Scan
                             //    1 = Scan erlaubt
      * int;                 // Makrofunktion
      * int;                 // Polygonfunktion
      * int;                 // Leiterbahnfunktion
      * int;                 // Textfunktion
      * int;                 // Bohrungsfunktion
      * int;                 // Lagencheckfunktion
      * int;                 // Levelcheckfunktion
      );
```

Beschreibung

Die Funktion **lay_scanfelem** scannt das angegebene Figurenelement über alle Hierarchiestufen. Dabei werden für alle gefundenen Elemente die referenzierten Anwenderfunktionen aufgerufen. Soll ein Funktionstyp nicht aufgerufen werden, so ist für den entsprechenden Parameter `NULL` anzugeben (Definition der referenzierten Anwenderfunktionen siehe **lay_scanall**). Der Rückgabewert der Funktion **lay_scanfelem** ist ungleich Null, wenn ungültige Parameter angegeben wurden, oder wenn eine der referenzierten Anwenderfunktionen einen Fehler an die Funktion **lay_scanfelem** zurückgemeldet hat.

Siehe auch

Funktionen **lay_maccoords**, **lay_scanall**, **lay_scanpool**.

lay_scanpool - Layout Scan über Poolelement (LAY)

Synopsis

```
int lay_scanpool(              // Scan Status
      void;                    // Poolelement
      double;                  // X-Offset (STD2)
      double;                  // Y-Offset (STD2)
      double;                  // Drehwinkel (STD3)
      int [0,1];               // Element in Arbeitsbereich Flag (STD10)
      int [0,1];               // Connectivity Scan Flag:
                               //    0 = kein Scan
                               //    1 = Scan erlaubt
      * int;                   // Makrofunktion
      * int;                   // Polygonfunktion
      * int;                   // Leiterbahnfunktion
      * int;                   // Textfunktion
      * int;                   // Bohrungsfunktion
      * int;                   // Lagencheckfunktion
      * int;                   // Levelcheckfunktion
      );
```

Beschreibung

Die Funktion **lay_scanpool** scannt das angegebene Poolelement über alle Hierarchiestufen. Dabei werden für alle gefundenen Elemente die referenzierten Anwenderfunktionen aufgerufen. Soll ein Funktionstyp nicht aufgerufen werden, so ist für den entsprechenden Parameter `NULL` anzugeben (Definition der referenzierten Anwenderfunktionen siehe **lay_scanall**). Der Rückgabewert der Funktion **lay_scanpool** ist ungleich Null, wenn ungültige Parameter angegeben wurden, oder wenn eine der referenzierten Anwenderfunktionen einen Fehler an die Funktion **lay_scanpool** zurückgemeldet hat.

Siehe auch

Funktionen **lay_maccoords**, **lay_scanall**, **lay_scanfelem**.

lay_setfigcache - Layout-Cache für den schnellen Zugriff auf Figurenlistenelemente aufbauen (LAY)

Synopsis

```
void lay_setfigcache(
      );
```

Beschreibung

Die Funktion **lay_setfigcache** baut den Layout-Cache für schnelle Figurenlistenelementzugriffe auf.

lay_setplanchkparam - Layout DRC Parameter setzen (LAY)

Synopsis

```
int lay_setplanchkparam(      // Status
      double ]0.0,[;          // Abstand Bahn zu Bahn (STD2)
      double ]0.0,[;          // Abstand Bahn zu Kupfer (STD2)
      double ]0.0,[;          // Abstand Kupfer zu Kupfer (STD2)
      double;                 // Standard-Leiterbahnbreite (STD2)
      string;                 // Blockname
      int [-6,99];            // Signallagencode (LAY1)
                              //  (Lage!=(-1) nur in BAE HighEnd zulässig)
      int [0,0[;              // DRC-Blocknummer
      );
```

Beschreibung

Die Funktion **lay_setplanchkparam** setzt Abstandsparameter für den Design Rule Check (DRC) im **Layouteditor**. Der Rückgabewert ist ungleich Null, wenn ungültige Abstände spezifiziert wurden. In **BAE Professional**, **BAE Economy** und **BAE Light** können nur die globalen Parameter für den Lagencode -1 (Alle Lagen) und den DRC-Block 0 gesetzt werden. In **BAE HighEnd** können darüber hinaus auch Parameterblöcke zur Zuweisung lagenspezifischer Mindestabstände für beliebige Signallagen (Lagencodes 0 bis 99), die oberste Lage (Lagencode -5) sowie die Innenlagen (Lagencode -6) spezifiziert werden. Der Parameterblock 0 ist immer vorhanden und enthält die global definierten DRC-Parameter. Ebenso ist auch immer für den Lagencode -1 (Alle Lagen) eine Vorgabe vorhanden.

Siehe auch

Funktion **lay_getplanchkparam**.

lay_toplayname - Layout Setup Name oberste Lage abfragen (LAY)

Synopsis

```
string lay_toplayname(        // Lagenname
      );
```

Beschreibung

Der Rückgabewert der Funktion **lay_toplayname** entspricht dem in der Setupdatei eingestellten Namen für die oberste Lage.

lay_vecttext - Layout Text vektorisieren (LAY)

Synopsis

```
int lay_vecttext(              // Status
      double;                  // X-Koordinate (STD2)
      double;                  // Y-Koordinate (STD2)
      double;                  // Drehwinkel (STD3)
      int [0,1];               // Spiegelung (STD14)
      double ]0.0,[;           // Text Größe (STD2)
      int [0,1];               // Physical Flag:
                               //    0 = Logical
                               //    1 = Physical
      int [0,2];               // Lagenspiegelung:
                               //    0 = Spiegelung aus
                               //    1 = X-Spiegelung
                               //    2 = Y-Spiegelung
      int [0,[;                // Text Stil (LAY14)
      string;                  // Text Zeichenkette
      * int;                   // Vektorisierungsfunktion
      );
```

Beschreibung

Die Funktion **lay_vecttext** vektorisiert den übergebenen Text unter Verwendung des aktuell geladenen Zeichensatzes. Dazu wird für jedes Textsegment die übergebene Vektorisierungsfunktion aufgerufen. Der Rückgabewert dieser Funktion ist ungleich Null, wenn ungültige Parameter angegeben wurden oder die vom Benutzer definierte Vektorisierungsfunktion einen Wert ungleich Null zurückgegeben hat.

Vektorisierungsfunktion

```
int vecfuncname(
      double x1,               // X-Koordinate erster Punkt (STD2)
      double y1,               // Y-Koordinate erster Punkt (STD2)
      double x2,               // X-Koordinate zweiter Punkt (STD2)
      double y2                // Y-Koordinate zweiter Punkt (STD2)
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Der Rückgabewert dieser Funktion sollte ungleich Null sein, wenn ein Fehler aufgetreten ist.

C.4.2 Layouteditor-Funktionen

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp GED zugeordnet, d.h. diese Funktionen können im **Layouteditor** aufgerufen werden:

ged_asklayer - GED Lagenauswahl (GED)

Synopsis

```
int ged_asklayer(              // Status
      & int;                   // Lagenrückgabe (LAY1|LAY9)
      int [0,7];               // Lagenabfragetyp:
                               //    0 = Dokumentarlagen und Signallagen
                               //    1 = Signallagen
                               //    2 = Signallagen
                               //        (inklusive Oberste Lage und Alle Lagen)
                               //    3 = Dokumentarlagen
                               //    4 = Signallagen und Versorgungslagen
                               //    5 = beliebige Anzeigeelementtypen
                               //    6 = Versorgungslagen
                               //    7 = Dokumentar-, Signal- und Versorgungslagen
      );
```

Beschreibung

Die Funktion **ged_asklayer** aktiviert im **Layouteditor** ein Lagenauswahlmenü. Der Lagenabfragetyp gibt an, welche Lagen bzw. Anzeigelementtypen zur Auswahl angeboten werden. Der Rückgabewert ist Null bei erfolgter Lagenwahl oder (-1) bei Wahl des Menüpunktes Abbruch.

ged_askrefname - Interaktive GED Referenznamensabfrage (GED)

Synopsis

```
int ged_askrefname(            // Status
      & string;                // Rückgabe Referenzname
      & index L_CPART;         // Rückgabe Netzlisten-Bauteilindex (nur auf
Layoutebene)
      int [0,2];               // Bauteilauswahlmodus:
                               //    0 = Alle Bauteile
                               //    1 = in aktueller Gruppe enthaltene Bauteile
                               //    2 = nicht in aktueller Gruppe enthaltene
Bauteile
      int [0,1];               // Flag - Auswahl nicht platzierter Bauteile
      );
```

Beschreibung

Die Funktion **ged_askrefname** aktiviert einen Dialog zur Referenzauswahl, d.h. zur Auswahl von Bauteilen auf Layoutebene bzw. zur Auswahl von Pins auf Bauteilebene. Über den Bauteilauswahlmodus und das Flag zur Selektion nicht platzierter Bauteile kann die zur Auswahl angebotene Bauteil- bzw. Pinliste eingeschränkt werden. Der Funktionsrückgabewert ist Null bei erfolgreicher Referenzauswahl oder ungleich Null wenn keine Referenzauswahl durchgeführt wurde.

ged_asktreeidx - GED Netzauswahlmenü aktivieren (GED)

Synopsis

```
int ged_asktreeidx(              // Status
      & string;                  // Rückgabe Netzname (nur auf Layoutebene)
      & index L_CNET;            // Rückgabe Netzindex (nur auf Layoutebene)
      int [0,5];                 // Netzauswahlmodus:
                                 //    0 = Alle Netze, inklusive
                                 //         Schaltfläche Keine Netzvorgabe
                                 //    1 = Sichtbare Netze
                                 //    2 = Unsichtbare Netze
                                 //    3 = Alle Netze
                                 //    4 = Direktsprung zu Netzpick
                                 //    5 = Alle Bäume, Namensmustereingabe zulässig
      );
```

Beschreibung

Die Funktion **ged_asktreeidx** aktiviert einen Dialog zur Netzauswahl. Über den Netzauswahlmodus kann die zur Auswahl angebotene Liste der Netze eingeschränkt werden. Der Funktionsrückgabewert ist Null bei erfolgreicher Netzauswahl, 1 wenn die Löschung eines Netzes (und damit keine Netzzuweisung) vorgenommen wurde, 2 wenn ein Netznamensmuster spezifiziert wurde, 3 bei Netznamensselektion per Netzklick oder ungleich Null bei ungültigen Parametern bzw. Abbruch des Auswahldialogs.

ged_attachtextpos - Textverschiebung an Layoutelement anfügen (GED)

Synopsis

```
int ged_attachtextpos(           // Status
      index L_FIGURE;            // Layoutelement
      string;                    // Text
      int;                       // Textlage (LAY1|LAY9)
      double;                    // Text-X-Koordinate (STD2)
      double;                    // Text-Y-Koordinate (STD2)
      double;                    // Textdrehwinkel (STD3)
      double;                    // Textgröße (STD2; negativ für Textbasislinie)
      int [0,1];                 // Textspiegelungsmodus (STD14)
      );
```

Beschreibung

Die Funktion **ged_attachtextpos** weist die Parameter für Lage, Position, Drehwinkel, Größe und Spiegelung an den angegebenen Text des spezifierten Layoutelements zu. Der Funktionsrückgabewert ist Null bei erfolgreicher Zuweisung, (-1) bei ungültigen Parametern oder (-2) wenn das Layoutelement keine Definition für den angegebenen Text enthält.

Siehe auch

Funktion **ged_storetext**.

ged_delelem - GED Element löschen (GED)

Synopsis

```
int ged_delelem(                // Status
      & index L_FIGURE;         // Element
      );
```

Beschreibung

Die Funktion **ged_delelem** löscht das übergebene Element aus der Elementliste. Der Rückgabewert ist Null bei erfolgter Löschung und (-1), wenn das übergebene Element ungültig ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

Siehe auch

Funktion **ged_drawelem**.

ged_drawelem - GED Elementanzeige aktualisieren (GED)

Synopsis

```
void ged_drawelem(
      index L_FIGURE;           // Element
      int [0, 4];               // Zeichenmodus (STD19)
      );
```

Beschreibung

Die Funktion **ged_drawelem** aktualisiert die Anzeige des angegebenen Elements unter Verwendung des spezifizierten Zeichenmodus.

Siehe auch

Funktion **ged_delelem**.

ged_drcerrorhide - GED DRC-Fehlerakzeptanzmodus setzen/rücksetzen (GED)

Synopsis

```
int ged_drcerrorhide(           // Status
      string;                   // Fehler-Id-String
      int;                      // Flag - Fehler ausblenden
      );
```

Beschreibung

Die Funktion **ged_drcerrorhide** setzt den Anzeigemodus für den über die Fehler-Id spezifizierten DRC-Fehler. Der Funktionsrückgabewert ist Null, wenn der Anzeigemodus erfolgreich gesetzt wurde, oder ungleich Null im Fehlerfall.

ged_drcpath - GED Designregelprüfung für Leiterbahn-Testplatzierung (GED)

Synopsis

```
int ged_drcpath(                 // Status
      int [0,99];                // Leiterbahnlage (LAY1)
      double ]0.0,[;             // Leiterbahnbreite (STD2)
      index L_LEVEL;             // Leiterbahn-Signalnetznummer/-Level
      int [0,3];                 // Leiterbahn-Connectivity-Prüfmodus:
                                 //    0 = Uneingeschränkte Connectivity-Prüfung
                                 //    1 = Abstandsprüfung gegen
                                 //         Leiterbahnsignalnetzbaum unterdrücken
                                 //    2 = Abstandsverletzungen zu allen
                                 //         Verbindungsbäumen anzeigen
                                 //    3 = Abstandsverletzungen zu allen
                                 //         Verbindungsbäumen neben Pickelement
anzeigen
      );
```

Beschreibung

Die Funktion **ged_drcpath** führt eine Designregelprüfung für eine Leiterbahngenerierung mit den angegebenen Parametern durch, ohne die Leiterbahn tatsächlich zu erzeugen. Die Koordinaten für die Leiterbahngenerierung werden aus der mit **bae_storepoint** aktuell erzeugten internen Polygonpunktliste entnommen. Der Funktionsrückgabewert ist Null, wenn die Leiterbahn ohne Designregelverletzung platziert werden kann, einen Wert grössr gleich (1) wenn die Leiterbahnplatzierung eine Designregelverletzung verursachen würde, oder (-1) bei fehlenden oder falschen Parametern bzw. bei unültiger Interpreterumgebung.

Siehe auch

Funktionen **bae_storepoint**, **ged_storepoint**.

ged_drcpoly - GED Designregelprüfung für Polygon-Testplatzierung (GED)

Synopsis

```
int ged_drcpoly(                 // Status
      int;                       // Polygonlage (LAY1)
      int [1,9];                 // Polygontyp (LAY4)
      string;                    // Polygonnetzname (für LAY4 types 4, 6 and 9)
      index L_LEVEL;             // Polygon-Signalnetznummer/-Level
      int [0,3];                 // Polygon-Connectivity-Prüfmodus:
                                 //    0 = Uneingeschränkte Connectivity-Prüfung
                                 //    1 = Abstandsprüfung gegen
                                 //         Polygonsignalnetzbaum unterdrücken
                                 //    2 = Abstandsverletzungen zu allen
                                 //         Verbindungsbäumen anzeigen
                                 //    3 = Abstandsverletzungen zu allen
                                 //         Verbindungsbäumen neben Pickelement
anzeigen
      );
```

Beschreibung

Die Funktion **ged_drcpoly** führt eine Designregelprüfung für eine Flächengenerierung mit den angegebenen Parametern durch, ohne die Fläche tatsächlich zu erzeugen. Die Koordinaten für die Polygongenerierung werden aus der mit **bae_storepoint** aktuell erzeugten internen Polygonpunktliste entnommen. Der Funktionsrückgabewert ist Null, wenn das Polygon ohne Designregelverletzung platziert werden kann, einen Wert grösser gleich (1) wenn die Polygonplatzierung eine Designregelverletzung verursachen würde, oder (-1) bei fehlenden oder falschen Parametern bzw. bei unültiger Interpreterumgebung.

Siehe auch

Funktionen **bae_storepoint**, **ged_storepoly**.

ged_drcvia - GED Designregelprüfung für Via-Testplatzierung (GED)

Synopsis

```
int ged_drcvia(                 // Status
      string;                   // Via Padstack-Bibliothekselementname
      double;                   // Via-X-Koordinate (STD2)
      double;                   // Via-Y-Koordinate (STD2)
      index L_LEVEL;            // Via-Signalnetznummer/-Level
      int [0,3];                // Via-Connectivity-Prüfmodus:
                                //    0 = Uneingeschränkte Connectivity-Prüfung
                                //    1 = Abstandsprüfung gegen
                                //          Viasignalnetzbaum unterdrücken
                                //    2 = Abstandsverletzungen zu allen
                                //          Verbindungsbäumen anzeigen
                                //    3 = Abstandsverletzungen zu allen
                                //          Verbindungsbäumen neben Pickelement
anzeigen
      );
```

Beschreibung

Die Funktion **ged_drcvia** führt eine Designregelprüfung für die Viaplatzierung mit den angegebenen Parametern durch, ohne das Via tatsächlich zu platzieren. Der Rückgabewert ist Null, wenn das Via ohne Designregelverletzung platziert werden kann, einen Wert grösser gleich (1) wenn die Viaplatzierung eine Designregelverletzung verursachen würde, (-1) bei fehlenden oder falschen Parametern bzw. bei unültiger Interpreterumgebung, oder (-2) wenn das angeforderte Padstacksymbol nicht verfügbar ist.

Siehe auch

Funktion **ged_storeuref**.

ged_elemangchg - GED Elementwinkel ändern (GED)

Synopsis

```
int ged_elemangchg(             // Status
      & index L_FIGURE;         // Element
      double;                   // Neuer Winkel (STD3)
      );
```

Beschreibung

Die Funktion **ged_elemangchg** ändert den Drehwinkel des übergebenen Elements. Der Drehwinkel wird ausgehend vom Nullwinkel eingestellt, d.h. der vorhergehende Drehwinkel des Elements hat keinen Einfluss auf das Ergebnis. Die Winkelangabe wird als Bogenmaßwert interpretiert. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht drehbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ged_elemfixchg - GED Element fixiert-Flag ändern (GED)

Synopsis

```
int ged_elemfixchg(             // Status
      & index L_FIGURE;         // Element
      int [0,1];                // Neues fixiert Flag (STD11)
      );
```

Beschreibung

Die Funktion **ged_elemfixchg** ändert den Fixiert-Modus des übergebenen Elements. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht fixierbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in **forall**-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ged_elemgrpchg - GED Element Gruppenflag ändern (GED)

Synopsis

```
int ged_elemgrpchg(             // Status
      index L_FIGURE;           // Element
      int [0,6];                // Neue Gruppenzugehörigkeit (STD13|0x4)
      );
```

Beschreibung

Die Funktion **ged_elemgrpchg** ändert die Gruppenzugehörigkeit des übergebenen Elements. Durch Setzen des Bits mit der Wertigkeit 3 (0x4) im Gruppenstatusparameter kann ein Meldungszeilenreport über das selektierte/deselektierte Element und die Gesamtzahl der in der Gruppe befindlichen Elemente aktiviert werden. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig ist oder (-2) wenn es sich nicht um ein gruppenselektierbares Element handelt. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

ged_elemlaychg - GED Elementlage ändern (GED)

Synopsis

```
int ged_elemlaychg(             // Status
      & index L_FIGURE;         // Element
      int;                      // Neue Lage (LAY1)
      );
```

Beschreibung

Die Funktion **ged_elemlaychg** ändert die Lagenzugehörigkeit des übergebenen Elements. Bei Bohrungen gibt die Lage die Bohrungsklasse an. Die Lage kann für Flächen, Leiterbahnen, Bohrungen, Texte und Pads auf Padstackebene geändert werden. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn die Lage nicht änderbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in **forall**-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ged_elemmirrchg - GED Elementspiegelung ändern (GED)

Synopsis

```
int ged_elemmirrchg(            // Status
      & index L_FIGURE;         // Element
      int [0,18];               // Neuer Spiegelungsmodus (STD14|LAY3)
      );
```

Beschreibung

Die Funktion **ged_elemmirrchg** ändert den Spiegelungsmodus des übergebenen Elements. Der Spiegelungsmodus kann bei Flächen, Texten und Referenzen geändert werden. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig ist oder (-2) wenn es keinen Spiegelungsmodus besitzt. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ged_elemposchg - GED Elementposition ändern (GED)

Synopsis

```
int ged_elemposchg(             // Status
      & index L_FIGURE;         // Element
      double;                   // X-Position (STD2)
      double;                   // Y-Position (STD2)
      );
```

Beschreibung

Die Funktion **ged_elemposchg** ändert die Position des übergebenen Elements. Bei Flächen/Leiterbahnen wird die Fläche/Leiterbahn so verschoben, dass der erste Punkt der Fläche/Leiterbahn auf der angegebenen Position zu liegen kommt. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht positionierbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ged_elemsizechg - GED Elementgröße ändern (GED)

Synopsis

```
int ged_elemsizechg(            // Status
      & index L_FIGURE;         // Element
      double;                   // Neue Größe (STD2)
      );
```

Beschreibung

Die Funktion **ged_elemsizechg** ändert die Größe des übergebenen Elements. Eine Größenänderung ist bei Texten, Bohrungen, Leiterbahnen und Flächen möglich. Bei Leiterbahnen wird mit der Größe die Leiterbahnbreite spezifiziert. Bei Flächen wird mit der Größe die Expansionsdistanz definiert. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht größenveränderbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ged_getautocornins - GED Modus für automatische Eckpunktgenerierung abfragen (GED)

Synopsis

```
int ged_getautocornins(       // Rückgabe Modus:
                              //    0 = Automatische Eckpunktgenerierung
                              //         deaktiviert
                              //    1 = Automatische Eckpunktgenerierung
                              //         für Leiterbahnen
                              //    2 = Automatische Eckpunktgenerierung
                              //         für Polygone
                              //    3 = Automatische Eckpunktgenerierung
                              //         für Leiterbahnen und Polygone
      );
```

Beschreibung

Der Rückgabewert der Funktion **ged_getautocornins** entspricht dem im **Layouteditor** aktuell eingestellten Modus für das automatische Einfügen von Ecken bei der Generierung von Leiterbahnen und Polygonen. Der Eckpunktgenerierungsmodus wird im **Layouteditor** über eine der Optionen Winkel+Raster oktagonal bzw. Nur Winkel oktagonal der Funktion Eingaberaster selektiert.

ged_getdblpar - GED Doubleparameter abfragen (GED)

Synopsis

```
int ged_getdblpar(            // Status
      int [0,[;               // Parametertyp/-nummer:
                              //    0 = X-Koordinate letzte Gruppenplatzierung
                              //         (STD2)
                              //    1 = Y-Koordinate letzte Gruppenplatzierung
                              //         (STD2)
                              //    2 = Standardbauteilplatzierungswinkel (STD3)
                              //    3 = Flächenautomatik Isolationsabstand (STD2)
                              //    4 = Flächenautomatik Min. Strukturgröße (STD2)
                              //    5 = Flächenautomatik Wärmefallenbreite (STD2)
                              //    6 = Flächenautomatik Wärmefallenisolation
                              //         (STD2)
                              //    7 = Flächenautomatik Schraffurlinienabstand
                              //         (STD2)
                              //    8 = Flächenautomatik Schraffurlinienbreite
                              //         (STD2)
                              //    9 = Flächenautomatik Schraffurlinienwinkel
                              //         (STD3)
                              //    10 = Netzsichtbarkeitsdialog
                              //          Netznamenskontollelementbreite
                              //    11 = Standardtextgröße (STD2)
                              //    12 = DRC-Abstandshaltung Textgröszlig;e (STD2)
                              //    13 = Autoplacement Bauteilexpansion (STD2)
                              //    14 = Autoplacement Bauteilpinfaktor [0, 1.0]
                              //    15 = Autoplacement Segmentpassung [0, 1.0]
                              //    16 = Autoplacement Bauteilkonturoffset (STD2)
                              //    17 = Standardtextplatzierungswinkel (STD3)
                              //    18 = Autorouter Umrandungsabstand (STD2)
                              //    19 = Autorouter Abstand Wärmefalle zu
                              //          Bohrung (STD2)
                              //    20 = Autorouter Abstand Isolation zu
                              //          Bohrung (STD2)
                              //    21 = Autorouter Maximale
                              //          Versorgungslänge (STD2)
                              //    22 = Autorouter Angefordertes
                              //          Spezial Routing-Raster (STD2)
                              //    23 = Autorouter Split Power Plane
                              //          Abstandshaltung (STD2)
                              //    24 = Autorouter BGA-Rastertoleranz (STD2)
                              //    25 = Autorouter Maximale
                              //          SMD-Fanoutlänge (STD2)
                              //    26 = Autorouter Pin-Via-Mindestabstand (STD2)
                              //    27 = CAM Gerber Standardlinienbreite (STD2)
                              //    28 = CAM Minimaler Abstand Wärmefalle
                              //          zu Bohrung (STD2)
                              //    29 = CAM Minimaler Abstand Isolation
                              //          zu Bohrung (STD2)
                              //    30 = CAM Toleranz Abstand Wärmefalle
                              //          zu Bohrung (STD2)
                              //    31 = CAM Toleranz Abstand Isolation
                              //          zu Bohrung (STD2)
                              //    32 = CAM Breite Versorgungslagenumrandung
                              //          (STD2)
                              //    33 = CAM Versorgungslagen-Isolationsabstand
                              //          (STD2)
                              //    34 = Busbahnbreite (STD2)
                              //    35 = Busbahnabstand (STD2)
      & double;               // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **ged_getdblpar** dient der Abfrage von mit **ged_setdblpar** im **Layouteditor** gesetzten Parametern vom Typ `double`. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **ged_getintpar**, **ged_getstrpar**, **ged_setdblpar**, **ged_setintpar**, **ged_setstrpar**.

ged_getdrcmarkmode - GED DRC Fehleranzeigemodus abfragen (GED)

Synopsis

```
int ged_getdrcmarkmode(       // DRC-Fehleranzeigemodus
      );
```

Beschreibung

Die Funktion **ged_getdrcmarkmode** dient der Abfrage des im **Layouteditor** aktuell aktiven DRC-Fehleranzeigemodus. Der Funktionsrückgabewert ist Null, wenn die selektierte Farbe für Fehler zur Fehlermarkeranzeige verwendet wird. Bei Verwendung der Highlightfarbe zur Fehlermarkeranzeige wird der Wert 1 zurückgegeben.

Siehe auch

Funktion **ged_setdrcmarkmode**.

ged_getdrcstatus - GED DRC Vollständigkeitsstatus abfragen (GED)

Synopsis

```
int ged_getdrcstatus(         // DRC-Status
      );
```

Beschreibung

Mit der Funktion **ged_getdrcstatus** kann der Vollständigkeitsstatus der Designregelprüfung im **Layouteditor** abgefragt werden. Der Funktionsrückgabewert ist Null, wenn lediglich die in der aktuellen Programmsitzung durchgeführten Designänderungen geprüft wurden. Wenn hingegen für das aktuell geladene Element eine vollständige Designregelprüfung durchgeführt wurde, und somit sämtliche Designregelverletzungen angezeigt werden, dann wird ein Wert ungleich Null zurückgegeben.

ged_getgroupdata - GED Gruppenplatzierungsdaten abfragen (GED)

Synopsis

```
int ged_getgroupdata(         // Status
      & double;               // Gruppenbezugspunkt X-Koordinate (STD2)
      & double;               // Gruppenbezugspunkt Y-Koordinate (STD2)
      & double;               // Gruppendrehwinkel (STD3)
      & double;               // Gruppenskalierungsfaktor
      & int;                  // Gruppenspiegelungsmodus
      & double;               // Gruppenquadrant X-Koordinate (STD2)
      & double;               // Gruppenquadrant Y-Koordinate (STD2)
      & int;                  // Gruppenquadrantmodus
      & int;                  // Gruppenbereichsmodus
      );
```

Beschreibung

Die Funktion **ged_getgroupdata** ermöglicht die Abfrage der aktuellen Eingabedaten während interaktiver Gruppenplatzierungen im **Layouteditor**. Der Funktionsrückgabewert ist ungleich Null wenn gerade keine Gruppenplatzierungsinteraktion aktiviert ist.

Siehe auch

Funktion **ged_getinputdata**.

ged_gethighlnet - GED Netz Highlightmodus/Farbe abfragen (GED)

Synopsis

```
int ged_gethighlnet(            // Status
      int [-1,[;                // Netznummer oder -1 für Highlightfokusmodusabfrage
      & int;                    // Highlightmodus
      & int;                    // Highlightfarbe (Bit 1 bis 6, STD18) und
                                //  Highlightmuster (Bit 7 bis 12)
      );
```

Beschreibung

Mit der Funktion **ged_gethighlnet** kann der Highlightmodus sowie die Highlightfarbe und das Highlightmuster für das Netz mit der angegebenen Netznummer abgefragt werden. Bei aktiviertem Netzhighlight wird im Parameter für den Highlightmodus ein Wert ungleich Null zurückgegegen, bei deaktiviertem Netzhighlight ergibt sich der Highlightmodusparameter zu Null. Im zweiten Parameter wird der Farbcode (Bit 1 bis 6) und das Muster (Bit 7 bis 12) für die Highlightanzeige zurückgegeben. Der Funktionsrückgabewert ergibt sich zu einem Wert ungleich Null, wenn die Abfrage erfolgreich war, andernfalls (Netz nicht gefunden, ungültige Parameter) wird der Wert Null zurückgegeben.

Siehe auch

Funktion **ged_highlnet**.

ged_getinputdata - GED Eingabedaten abfragen (GED)

Synopsis

```
int ged_getinputdata(           // Status
      & double;                 // Ursprüngliche X-Koordinate (STD2)
      & double;                 // Ursprüngliche Y-Koordinate (STD2)
      & double;                 // Ursprüngliche Breite (STD2)
      & int;                    // Ursprüngliche Lage (LAY1)
      & double;                 // Aktuelle X-Koordinate (STD2)
      & double;                 // Aktuelle Y-Koordinate (STD2)
      & double;                 // Aktuelle Breite (STD2)
      & int;                    // Aktuelle Lage (LAY1)
      & void;                   // Eingabemodus/Element (LAY11)
      & void;                   // Leiterbahn Netzindex (optional)
      & double;                 // Erstes Segment Start-X-Koordinate (STD2) */
      & double;                 // Erstes Segment Start-Y-Koordinate (STD2) */
      & double;                 // Erster Kreismittelpunkt X-Koordinate (STD2) */
      & double;                 // Erster Kreismittelpunkt Y-Koordinate (STD2) */
      & int;                    // Erster Kreismittelpunkt Typ (STD15) */
      & double;                 // Letztes Segment Start-X-Koordinate (STD2) */
      & double;                 // Letztes Segment Start-Y-Koordinate (STD2) */
      & double;                 // Letzter Kreismittelpunkt X-Koordinate (STD2) */
      & double;                 // Letzter Kreismittelpunkt Y-Koordinate (STD2) */
      & int;                    // Letzter Kreismittelpunkt Typ (STD15) */
      );
```

Beschreibung

Die Funktion **ged_getinputdata** ermöglicht die Abfrage der aktuellen Eingabedaten während interaktiver Platzierungen im **Layouteditor**. Die Interpretation der Platzierungsdaten ist entsprechend dem zurückgelieferten Funktionsparameter für den Eingabemodus bzw. das Platzierungselement vorzunehmen. Der Funktionsrückgabewert ist ungleich Null wenn gerade keine Platzierungsinteraktion aktiviert ist.

Siehe auch

Funktion **ged_getgroupdata**.

ged_getintpar - GED Integerparameter abfragen (GED)

Synopsis

```
int ged_getintpar(              // Status
      int [0,[;                 // Parametertyp/-nummer:
                                //    0 = Pickpunktanzeigemodus:
                                //        0 = keine Pickpunktanzeige
                                //        1 = Pickpunktanzeige
                                //        2 = Pickpunktübersichtsanzeige
                                //        3 = Pickpunkteditieranzeige
                                //    1 = automatischer DRC beim Laden:
                                //        0 = kein automatischer DRC
                                //        1 = automatischer DRC nach Abfrage
                                //        2 = automatischer DRC ohne Abfrage
                                //    2 = Farbcode für Oberste Lage
                                //    3 = Infoanzeigeflag:
                                //        0 = keine automatische Infoanzeige
                                //        1 = automatische Infoanzeige
                                //    4 = Infoanzeigemodus:
                                //        0 = keine Infoanzeige
                                //        1 = komplette Infoanzeige
                                //        2 = nur Kupferinfoanzeige
                                //    5 = Winkeleditierrichtung
                                //    6 = Bauteilmakro-DRC:
                                //        0 = kompletter DRC
                                //        1 = Bauteilmakros als geprüft betrachten
                                //    7 = Rastereckenabfragemodus:
                                //        0 = keine Rastereckenabfrage
                                //        1 = komplette Rastereckanabfrage
                                //        2 = Rastereckenabfrage im aktuellen
                                //             Fenster
                                //        3 = Rastereckenabfrage im dynamisch
                                //             aktualisierten Fenster
                                //    8 = Minconaktualisierungsmodus
                                //    9 = DRC Polygontyp-Abschaltungsbits
                                //    10 = Warnmeldemodus:
                                //        Bit 0: SCM-Änderungs-Warnungen unterdrückt
                                //        Bit 1: Warnungen Gruppenselektion
                                //               Füllflächenproblempolygone
                                //               unterdrückt
                                //        Bit 2: Variantenvergleichswarnungen
                                //               unterdrückt
                                //        Bit 3: Warnungen über Beendingung des
                                //               Autoroutermodus unterdrückt
                                //    11 = Lagenbenutzung Scanmodus
                                //    12 = Flächenpolygoneditiermodus:
                                //        0 = keine geschlossenen Linienzüge
                                //        1 = immer geschlossene Linienzüge
                                //        2 = Abfrage zum Schließen von Linienzügen
                                //    13 = DRC-Abstandsanzeigemuster
                                //    14 = Leiterbahneditiermodus:
                                //        0 = Eingaberasterfang
                                //        1 = Pin-/Segmentfang bei erster
                                //             Leiterbahnecke
                                //    15 = Flächenspiegelsicht:
                                //        0 = Standardflächenspiegelsicht
                                //        1 = Flächenspiegelsicht deaktiviert
                                //    16 = Bestätigungsabfragelimit für Netz löschen
                                //    17 = Plotvorschaumodus:
                                //        0 = keine Plotvorschau
                                //        1 = Plotterstiftbreite
                                //    18 = DRC-Distanzanzeigemodus:
                                //        0 = keine DRC-Distanzanzeige
                                //        1 = Leiterbahndistanzlinien
                                //        2 = Flächendistanzlinien
                                //        3 = Leiterbahndistanzmuster
                                //        4 = Flächendistanzmuster
```

```
                              //    19 = Textlagenspiegelungsmodus:
                              //        0 = keine Textlagenspiegelung
                              //        1 = Dokumentarlagenspiegelung
                              //        2 = Signal- und Dokumentarlagenspiegelung
                              //    20 = Standardbauteilspiegelungsmodus
                              //    21 = Autosave Intervall
                              //    22 = Bauteilluftlinienanzeigemodus:
                              //        0 = Keine Luftlinienanzeige
                              //        1 = Statische Luftlinienanzeige
                              //        2 = Dynamische Luftlinienanzeige
                              //    23 = Winkelfreigabeumschaltmodus:
                              //        0 = Umschaltung Pickseite standard
                              //        1 = Umschaltung Raster
                              //        2 = Umschaltung kürzere Seite
                              //        3 = Umschaltung Editierrichtung
                              //    24 = Copper fill heat trap mode
                              //        3 = Via-Wärmefallen
                              //        |4 = Flag - keine benachbarten Pins
                              //        |8 = Flag - Wärmefallenleiterbahnen
                              //        |16 = Flag - nur unverbundene Lagen
                              //    25 = Flächenautomatik Leiterbahnmodus
                              //        0 = Runde Ecken
                              //        1 = Oktagonale Exken
                              //        2 = Oktagonale Kreise
                              //        3 = Oktagonale Ecken & Kreise
                              //    26 = Flächenautomatik Inselerkennung:
                              //        0 = Inseln beibehalten
                              //        1 = Inseln löschen
                              //        2 = Inseln selektieren
                              //    27 = Flächenautomatik Innenflächenmodus:
                              //        0 = Innenfüllflächen füllen
                              //        1 = Innenfüllflächen ausparen
                              //        |2 = Ausparflächen mit Abstand
                              //    28 = Flächenautomatik
                              //          Wärmefallenverbingsanzahl
                              //    29 = Flächenautomatik Spitzwinkelmodus:
                              //        0 = Spitze Winkel flach
                              //        1 = Spitze Winkel rund
                              //    30 = Flächenautomatik Schraffurmodus:
                              //        0 = Linienschraffur
                              //        1 = Gitterschraffur
                              //        |2 = Editierbare Pfade erzeugen
                              //    31 = Netzsichtbarkeitsdialogmodus:
                              //        0 = Einzelspalte für
                              //            Netznamenslistenanzeige
                              //        1 = Mehrspaltige Netznamenslistenanzeige
                              //    32 = Gruppenbewegtdarstellung:
                              //        0 = Bewegtbild aus
                              //        1 = Nur Baugruppenlage
                              //        2 = Bewegtbild ein
                              //        3 = Bewegtbild alles
                              //    33 = Leiterbahngruppenselektionsmodus:
                              //        0 = Leiterbahn- & Viaselektion
                              //        1 = Leiterbahnselektion
                              //        2 = Viaselektion
                              //    34 = Vorzugslagenauswahl (LAY1)
                              //    35 = Zwischenablagetext-
                              //          Platzierungsanforderung
                              //    36 = Editierrichtung
                              //    37 = Mincon-Flächenmodus (Bitmuster):
                              //        0 = Kein Flächen-Mincon
                              //        |1 = Kupferflächen-Mincon
                              //        |2 = Potentialflächen-Mincon
                              //    38 = Gruppenwinkelfreigabemodus:
                              //        0 = Gruppenwinkel einhalten
                              //        1 = Gruppenwinkel automatisch freigeben
                              //    39 = Autoplacement Optimierungszahl
                              //    40 = Autoplacement Bauteiltausch
```

```
                                //     41 = Autoplacement Pin-/Gattertausch
                                //     42 = Autoplacement Spiegelungsmodus:
                                //         0 = Kein SMD-Spiegeln
                                //         1 = SMD-Spiegeln
                                //         2 = 2 Pin-SMD-Spiegeln
                                //         3 = Immer SMD-Spiegeln
                                //     43 = Autoplacement Rotationsmodus:
                                //         0 = 0-90 Grad Drehung
                                //         1 = 0-270 Grad Drehung
                                //         2 = 0 Grad Drehung
                                //         3 = 90 Grad Drehung
                                //         4 = 0 XOR 90 Grad Drehung
                                //     44 = Autoplacement Retrydurchgänge
                                //     45 = Autoplacement SMD-Rotationsmodus:
                                //         0 = 0-90 Grad Drehung
                                //         1 = 0-270 Grad Drehung
                                //         2 = 0 Grad Drehung
                                //         3 = 90 Grad Drehung
                                //         4 = 0 XOR 90 Grad Drehung
                                //     46 = Autoplacement Bauteilkonturlage (LAY1)
                                //     47 = Gruppenbearbeitung Elementfang:
                                //         0 = Alle Elemente selektieren
                                //         1 = Nur sichtbare Elemente selektieren
                                //     48 = Standardtextspiegelung und
                                //           Standardtextmodus (STD14|LAY14)
                                //     49 = Autorouter Anzahl Optimierungen
                                //     50 = Autorouter Optimierer Richtungsmodus
                                //     51 = Autorouter Max. Via-Zahl
                                //     52 = Autorouter Via-Kosten
                                //     53 = Autorouter Pinkanal-Kosten
                                //     54 = Autorouter Anti-Vorzugs-Richtungs-Kosten
                                //     55 = Autorouter Richtungsänderungs-Kosten
                                //     56 = Autorouter Packungs-Kosten
                                //     57 = Autorouter Dynamische-
                                //           Dichtestatistik-Basis
                                //     58 = Autorouter Max. Bäume pro Rip-Up
                                //     59 = Autorouter Max. Rip-Up Tiefe
                                //     60 = Autorouter Max. Rip-Up Wiederholungen
                                //     61 = Autorouter Via-Raster Index
                                //     62 = Autorouter Bus-Abknickkosten
                                //     63 = Autorouter Abstand-1-Kosten
                                //     64 = Autorouter Abstand-2-Kosten
                                //     65 = Autorouter Rip-Up-Kosten
                                //     66 = Autorouter Router Cleanup
                                //     67 = Autorouter Optimierer Cleanup
                                //     68 = Autorouter Versorgungsanschluss
                                //           Vektorunroutes
                                //     69 = Autorouter Zwischenspeichern
                                //     70 = Autorouter Verbindungseckenausgabemodus
                                //     71 = Autorouter Unroutesausgabesortiermodus
                                //     72 = Autorouter Ecken Spitzwinkel Modus
                                //     73 = Autorouter Richtungsmodus
                                //           existierende Bahnen
                                //     74 = Autorouter Standardverbindungs-
                                //           Lagen-Kosten
                                //     75 = Autorouter Busverbindungs-Lagen-Kosten
                                //     76 = Autorouter Offset Wellenausbreitung Limit
                                //     77 = Autorouter Via Raster Modus
                                //     78 = Autorouter Eingabedaten Fehlerprüfmodus
                                //     79 = Autorouter Pin Anschluss Modus
                                //     80 = Autorouter Halbraster-Modus
                                //     81 = Autorouter Off-Grid-Routing-Kosten
                                //     82 = Autorouter Buserkennung/Busrouting Modus
                                //     83 = Autorouter SMD Via-Vorlegen
                                //     84 = Autorouter Router Pin/Gate-Swap
                                //     85 = Autorouter Gridless Routing Modus
                                //     86 = Autorouter Inkrementaler Ausgabemodus
                                //     87 = Autorouter Vorzugsraster
                                //     88 = Autorouter Anti-Vorzugsraster-Kosten
                                //     89 = Autorouter Anti-Netzbereichs-Kosten
                                //     90 = Autorouter Letzter Optimiererparameter
```

```
                              //          Wechselmodus
                              //     91 = Autorouter Auto-Rip-Up Parameter Modus
                              //     92 = Autorouter Vorzugsrichtungsmodus
                              //     93 = Autorouter Optimierer Reihenfolge Modus
                              //     94 = Autorouter Via Rip-Up Flag
                              //     95 = Autorouter Routingfenster Randausdehnung
                              //     96 = Autorouter BGA Fan out
                              //     97 = Autorouter Pinentry Korrektur
                              //     98 = Autorouter Alternativer Viaversatz Modus
                              //     99 = Autorouter Voll-Viaauswertemodus
                              //     100 = Autorouter Micro Via Modus
                              //     101 = Autorouter Richtungsvorgabe
                              //           Maximalabweichung
                              //     102 = Autorouter Routingfenster Flag
                              //     103 = Autorouter Padentry Subgrid
                              //     104 = Autorouter Versorgungslagen Vias
                              //     105 = Autorouter Via Prüfmodus
                              //     106 = Autorouter Verbindungsanzahl große Netze
                              //     107 = Autorouter Autorouting aktiv Flag
                              //     108 = CAM Wärmefallenbasiswinkel
                              //     109 = Flag - Einzeleckenbearbeitung
                              //     110 = Flag - Größenänderung mit Eckenabrundung
                              //     111 = Flag - Anzeige ausgeblendeter DRC-Fehler
                              //     112 = Flag - DRC-Verletzung Elementscan
                              //     113 = Bauteilplatzierung -
                              //           Bahnendenbewegungsmodus:
                              //         0 = Keine Bahnendenbewegung
                              //         1 = Bahnenden mitbewegen
                              //         2 = Bahnsegmente mitbewegen
                              //     114 = Flag - Polygonbearbeitung
                              //           Autocomplete-Modus
                              //     115 = Platinenumrandung alternative
                              //           Dokumentarlage (LAY1)
                              //     116 = Leiterbahnzusammenfassungsmodus:
                              //         0 = Bahnen nicht zusammenfassen
                              //         1 = Bahnen zusammenfassen
                              //         2 = Bahnzusammenfassungsabfrage
                              //     117 = Anzeigemodus Leiterbahnen (LAY15)
                              //     118 = Anzeigemodus Texte (LAY15)
                              //     119 = Anzeigemodus Kupferpolygone (LAY15)
                              //     120 = Anzeigemodus Sperrflächen (LAY15)
                              //     121 = Anzeigemodus Umrandungspolygon (LAY15)
                              //     122 = Anzeigemodus Potentialflächen (LAY15)
                              //     123 = Anzeigemodus Dokumentarlinien (LAY15)
                              //     124 = Anzeigemodus Dokumentarflächen (LAY15)
                              //     125 = Anzeigemodus Füllflächen mit
                              //           Ausschnittspolygonen (LAY15)
                              //     126 = Anzeigemodus Schraffurflächen (LAY15)
                              //     127 = Anzeigemodus Geteilte
                              //           Potentialflächen (LAY15)
                              //     128 = Flag - Farbtabelle gesichert
                              //     129 = Airlinefarbmodus:
                              //         0 = Airlinefarbe benutzen
                              //         1 = Lagenfarbe benutzen
                              //     130 = Airlineclippingmodus:
                              //         0 = Kein Airlineclipping
                              //         1 = Airlines ohne Zielpunkt im
                              //             Arbeitsbereich ausblenden
                              //     131 = Bahnkollisionsmodus:
                              //        -1 = Operationsabfrage
                              //         0 = Kollisionen ignorieren
                              //         1 = Kollidierende Bahnen löschen
                              //         2 = Kollidierende Segmente löschen
                              //         3 = Kollidierende Segmente abschneiden
```

```
                              //    132 = Abfragemodus für Layoutbahnverbindungen:
                              //        0 = Layoutbahnen nie verbinden
                              //        1 = Layoutbahnen immer verbinden
                              //        2 = Verbindungsmodus abfragen
                              //    133 = Abfragemodus für
                              //          Bauteilbahnverbindungen:
                              //        0 = Bauteilbahnen nie verbinden
                              //        1 = Bauteilbahnen immer verbinden
                              //        2 = Verbindungsmodus abfragen
                              //    135 = Polygonanzeigemodus beim Bewegen von
                              //           Elementen:
                              //        0 = Umrandungsanzeige
                              //        1 = Füllanzeige
                              //    136 = Airlineanzeigemodus beim Bewegen von
                              //           Gruppen:
                              //        0 = Airlineanzeige aus
                              //        1 = Airlines für Gruppenbauteilpins
                              //             anzeigen
                              //    137 = Abstandsprüfmodus für Bahnkollisionen:
                              //        0 = DRC-Abstand für Kollisionscheck
                              //             benutzen
                              //        1 = Nur Kreuzungen als Kollisionen
                              //             interpretieren
                              //    138 = Pickmodus für Leiterbahnbündel:
                              //        0 = Kontinuierlicher Segmentpick
                              //        1 = Pick erstes und letztes Bündelsegment
                              //    139 = Einfüge-Pickmodus für
                              //           Leiterbahnsegmente:
                              //        0 = 3-Klick Selektion
                              //        1 = 2-Klick Selektion
                              //    140 = Bauteil-DRC:
                              //        0 = kein Bauteil-Online-DRC
                              //        1 = Bauteil-Online-DRC
                              //    141 = Flag - Optimierung Bohrwerkzeugtabelle
                              //    142 = Busbahnanzahl
                              //    143 = Busbahnanzahl editieren
                              //    144 = Generierungsmodus Busbahnen:
                              //        0 = Bahnbündel generieren
                              //        1 = Separate Bahnen generieren
                              //    145 = Busbahn-Eckenmodus:
                              //        0 = Gewinkelte Ecken generieren
                              //        1 = Abgerundete Ecken gnerieren
                              //    146 = Siebdrucklage (LAY1)
                              //    147 = Anzeigmodus Makroumrandungen:
                              //        0 = Keine Makroumrandungsanzeige
                              //        1 = Makroumrandungen für bewegte
                              //             Referenzen anzeigen
                              //        2 = Makroumrandungen anzeigen
       & int;                 // Rückgabe Parameterwert
       );
```

Beschreibung

Die Funktion **ged_getintpar** dient der Abfrage von mit **ged_setintpar** im **Layouteditor** gesetzten Integerparametern. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **ged_getdblpar**, **ged_setintpar**, **ged_getstrpar**, **ged_setdblpar**. **ged_setstrpar**.

ged_getlaydefmode - GED Defaultlagenmodus abfragen (GED)

Synopsis

```
int ged_getlaydefmode(          // Defaultlagenmodus:
                                //    0 = automatische Lagendefaultzuweisung
                                //         deaktivieren
                                //    1 = benutzte Eingabelage ist Lagendefault
                                //    2 = zuletzt benutzte Lage ist Lagendefault
      );
```

Beschreibung

Die Funktion **ged_getlaydefmode** gibt den aktuell im **Layouteditor** eingestellten Defaultlagenmodus zurück.

Siehe auch

Funktionen **ged_getlayerdefault**, **ged_setlaydefmode**, **ged_setlayerdefault**.

ged_getlayerdefault - GED Defaultlage abfragen (GED)

Synopsis

```
int ged_getlayerdefault(        // Lage (LAY1)
      );
```

Beschreibung

Die Funktion **ged_getlayerdefault** gibt die aktuell im **Layouteditor** eingestellte Defaultlage zurück.

Siehe auch

Funktionen **ged_getlaydefmode**, **ged_setlaydefmode**, **ged_setlayerdefault**.

ged_getmincon - GED Mincon-Funktion abfragen (GED)

Synopsis

```
int ged_getmincon(              // Mincon Funktionstyp (LAY10)
      );
```

Beschreibung

Der Rückgabewert der Funktion **ged_getmincon** entspricht dem im **Layouteditor** aktuell eingestellten Wert des Mincon-Modus für die Airlineanzeige (LAY10).

ged_getpathwidth - GED Bahnenstandardbreite abfragen (GED)

Synopsis

```
void ged_getpathwidth(
      & double;                 // Schmal Standardbreite (STD2)
      & double;                 // Breit Standardbreite (STD2)
      );
```

Beschreibung

Die Funktion **ged_getpathwidth** gibt in den beiden Parametern die Werte der aktuell im **Layouteditor** eingestellten Standardbreiten für schmale und breite Leiterbahnen zurück.

ged_getpickmode - GED Elementpickmodus abfragen (GED)

Synopsis

```
int ged_getpickmode(           // Elementpickmodus:
                               //    0 = Pick über Vorzugslage
                               //    1 = Pick mit Elementauswahl
                               //    2 = Pick exklusiv über Vorzugslage
      );
```

Beschreibung

Die Funktion **ged_getpickmode** dient der Abfrage des aktuell im **Layouteditor** ausgewählten Elementpickmodus.

Siehe auch

Function **ged_setpickmode**.

ged_getpickpreflay - GED Vorzugslage abfragen (GED)

Synopsis

```
int ged_getpickpreflay(        // Vorzugslage (LAY1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **ged_getpickpreflay** entspricht der Vorzugslage (LAY1) für Elementwahl im **Layouteditor**.

ged_getpowlayerrcnt - GED Versorgungslagenfehleranzahl abfragen (GED)

Synopsis

```
int ged_getpowlayerrcnt(       // Versorgungslagenfehleranzahl
      );
```

Beschreibung

Die Funktion **ged_getpowlayerrcnt** gibt die Anzahl der im **Layouteditor** erkannten Versorgungslagenfehler zurück.

ged_getsegmovmode - GED Leiterbahnsegmentbewegungsmodus abfragen (GED)

Synopsis

```
int ged_getsegmovmode(         // Leiterbahnsegmentbewegungsmodus:
                               //    0 = Mit Nachbarn bewegen
                               //    1 = Ohne Nachbarn bewegen
                               //    2 = Nachbarn anpassen
                               //    3 = Nachbarn ohne Durchkontaktierungen
                               //         anpassen
                               //    8 = Nur unmittelbare Nachbarn anpassen
                               //    |4 = Offene Leiterbahnenden folgen
                               //          Segmentbewegung
      );
```

Beschreibung

Die Funktion **ged_getsegmovmode** dient der Abrage des aktuell im **Layouteditor** eingestellten Leiterbahnsegmentbewegungsmodus.

Siehe auch

Funktion **ged_setsegmovmode**.

ged_getstrpar - GED Stringparameter abfragen (GED)

Synopsis

```
int ged_getstrpar(              // Status
      int [0,[;                 // Parametertyp/-nummer:
                                //    0 = Name zuletzt platziertes benanntes Element
                                //    1 = Name zuletzt platziertes
                                //         Bibliothekselement
                                //    2 = Zuletzt platzierter Textstring
                                //    3 = Standardbibliotheksname
                                //    4 = Nächster freier Name
                                //    5 = Bohrbauteil-/Bohrpinnamensbasis
                                //    6 = Bohrbauteilmakronamensmuster
                                //    7 = Bohrpinmakronamensmuster
                                //    8 = Eingabeaufforderung Ersatzstring
                                //    9 = Bibliothek zuletzt platziertes Makro
                                //    10 = Autosave path name
      & string;                 // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **ged_getstrpar** dient der Abfrage von im **Layouteditor** gesetzten Stringparametern. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **ged_getdblpar**, **ged_getintpar**, **ged_setdblpar**, **ged_setintpar**, **ged_setstrpar**.

ged_getviaoptmode - GED Leiterbahnviaoptimierungsmodus abfragen (GED)

Synopsis

```
int ged_getviaoptmode(          // Leiterbahnviaoptimierungsmodus:
                                //    0 = Viaoptimierung
                                //    1 = Vias beibehalten
      );
```

Beschreibung

Die Funktion **ged_getviaoptmode** dient der Abrage des aktuell im **Layouteditor** eingestellten Leiterbahnviaoptimierungsmodus.

Siehe auch

Funktion **ged_setviaoptmode**.

ged_getwidedraw - GED Breitendarstellung abfragen (GED)

Synopsis

```
double ged_getwidedraw(         // Breite (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **ged_getwidedraw** entspricht der Breite, ab der im **Layouteditor** Leiterbahnen auf dem Bildschirm in Flächendarstellung angezeigt werden.

ged_groupselect - GED Gruppenselektion (GED)

Synopsis

```
int ged_groupselect(          // Anzahl Änderungen oder (-1) bei Fehler
      int [0,9];              // Element Selektionstyp:
                              //    0 = Selektion nach Elementtyp
                              //    1 = Selektion nach Lage
                              //    2 = Selektion nach Fixiertflag
                              //    3 = Selektion nach Sichtbarkeit
                              //    4 = Selektion nach Lage/negiert
                              //    5 = Selektion nach Netznummer
                              //    6 = Selektion nach Netznummer/negiert
                              //    7 = Selektion Elemente mit Verbindung zu Netz
                              //    8 = Selektion Elemente ohne Verbindung zu Netz
                              //    9 = Selektion nach Elementpolygontyp
                              //    10 = Selektion nach Elementverankerungsmodus
      int;                    // Element Selektionswert entspr. Selektionstyp:
                              //    0 - Elementtyp (0|LAY6)
                              //    1,4 - Elementlage (LAY1)
                              //    2 - Element-Fixiertflag (STD11)
                              //    3 - Elementsichtbarkeit (0|1)
                              //    5,6 - Elementnetznummer
                              //    7,8 - Netznummer
                              //    9 - Elementpolygontyp (LAY4)
                              //    10 - Elementverankerungsmodus (STD11 | STD12)
      int [0,2];              // Neue Gruppenzugehörigkeit (STD13)
      );
```

Beschreibung

Die Funktion **ged_groupselect** ändert die Gruppenzugehörigkeit aller Element des spezifizierten Typs bzw. mit der spezifizierten Eigenschaft. Der Rückgabewert entspricht der Anzahl der durchgeführtem Änderungen oder dem Wert (-1) bei fehlerhaften bzw. inkompatiblen Parameterangaben. Der Selektionswert Null bei der Selektion nach dem Elementtyp kann dazu benutzt werden, Elemente *beliebigen* Typs auszuwählen.

Warnung

Interne Layoutelementtypen wie z.B. die Standardvia-Definition(en) sind von der Gruppen(de)selektion mit **ged_groupselect** ausgenommen, um ein versehentliches Löschen bzw. Ändern derartiger Elemente durch die anschließende Anwendung anderer Gruppenfunktionen zu verhindern.

ged_highlnet - GED Netz Highlightmodus/Farbe setzen (GED)

Synopsis

```
int ged_highlnet(             // Status
      int [-1,[;              // Netznummer
      int [0,[;               // Highlight aus/ein Flag || (Farb- und
Musterdefininition << 1)
      );
```

Beschreibung

Mit der Funktion **ged_highlnet** kann der Highlightmodus und die Highlightfarbe für das Netz mit der angegebenen Netznummer gesetzt werden. Zur Aktivierung des Netzhighlights ist das Bit 1 des Highlightparameters auf 1 zu setzen, die Deaktivierung erfolgt entsprechend durch Spezifikation des Werts Null im Bit 1 des Highlightparameters. Über die restlichen Bits des Highlightparameters kann der gewünschte Farbcode (Bits 2 bis 6) bzw. ein Muster (Bits 7 bis 12) für die Highlightanzeige spezifiziert werden. Der Funktionsrückgabewert ergibt sich zu einem Wert ungleich Null bei erfolgreicher Änderung des netzspezifischen Highlightmodus, andernfalls (Netz nicht gefunden, ungültige Parameter) wird der Wert Null zurückgegeben.

Siehe auch

Funktion **ged_gethighlnet**.

ged_layergrpchg - GED Gruppenselektion nach Lage (GED)

Synopsis

```
int ged_layergrpchg(           // Anzahl Elemente
      int [0,[;                // Lagennummer (LAY1)
      int [0,2];               // Neue Gruppenzugehörigkeit (STD13)
      );
```

Beschreibung

Die Funktion **ged_layergrpchg** ändert die Gruppenzugehörigkeit aller Elemente, die auf der angegebenen Lage platziert sind. Der Rückgabewert ist die Anzahl der (de)selektierten Elemente oder (-1) bei Fehler.

ged_partaltmacro - GED Bauteilnamegehäusetyp ändern (GED)

Synopsis

```
int ged_partaltmacro(          // Status
      string;                  // Bauteilname
      string;                  // Neuer Bauteil Bibiotheksteilname
      );
```

Beschreibung

Die Funktion **ged_partaltmacro** ändert den Gehäusetyp des angegebenen Bauteiles. Ein Rückgabewert von Null zeigt eine erfolgreiche Änderung an. Bei ungültigen Eingabedaten wird (-1) zurückgegeben, (-2) wenn der neue Gehäusetyp nicht alle für dieses Bauteil in der Netzliste verwendeten Pins enthält (Gehäuseänderung wird trotzdem durchgeführt), (-3) wenn das Bauteil nicht in der Netzliste vorhanden ist, (-4) wenn der neue Gehäusetyp für dieses Bauteil nicht erlaubt ist, (-5) wenn der neue Gehäusetyp nicht ladbar ist, (-6) wenn die Bauteildaten nicht in die Jobdatenbank kopiert werden konnten und (-7) wenn versucht wurde, in einem Programmlauf einen Bauteilgehäusetyp mehrfach umzuändern (z.B. **a** in **b** und anschließend **b** in **c**). Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion sollte nicht innerhalb von **L_CPART**-Index-Schleifen aufgerufen werden, da die vor dem Aufruf der Funktion belegten **L_CPART**-Indexvariablen anschließend ungültig sind.

ged_partnamechg - GED Bauteilname ändern (GED)

Synopsis

```
int ged_partnamechg(           // Status
      string;                  // Alter Name
      string;                  // Neuer Name
      );
```

Beschreibung

Die Funktion **ged_partnamechg** ändert den Namen des angegebenen Bauteiles. Ein Rückgabewert von Null zeigt eine erfolgreiche Änderung an. Bei ungültigen Eingabedaten wird (-1) zurückgegeben, (-2) wenn das Bauteil nicht platziert ist, (-4) wenn der neue Name schon definiert ist und (-5) wenn versucht wurde, in einem Programmlauf ein Bauteil mehrfach umzubenennen (z.B. **a** in **b** und anschließend **b** in **c**). Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden. Auf Bauteilebene kann **ged_partnamechg** zur Umbenennung von Pins verwendet werden.

Warnung

Diese Funktion verändert möglicherweise die Netzliste und erfordert ggf. einen anschließenden Backannotation-Lauf. Die Funktion sollte auch nicht innerhalb von **L_CPART**-Index-Schleifen aufgerufen werden, da die vor dem Aufruf der Funktion belegten **L_CPART**-Indexvariablen anschließend ungültig sind.

ged_pickanyelem - Beliebiges GED Element selektieren (GED)

Synopsis

```
int ged_pickanyelem(            // Status
      & index L_FIGURE;         // Rückgabe selektiertes Element
      int;                      // Pickelementtypmenge ((LAY6 außer 7)<<1 verodert)
      );
```

Beschreibung

Die Funktion **ged_pickanyelem** aktiviert eine Mausinteraktion zur Selektion eines Elements aus der angegebenen Pickelementtypmenge. Der Rückgabewert ist Null bei erfolgter Selektion und (-1) wenn an der Pickposition kein Element gefunden wurde.

Siehe auch

Funktion **ged_pickelem**.

ged_pickelem - GED Element selektieren (GED)

Synopsis

```
int ged_pickelem(               // Status
      & index L_FIGURE;         // Rückgabe selektiertes Element
      int [1,10];               // Elementtyp (LAY6 außer 7)
      );
```

Beschreibung

Mit der Funktion **ged_pickelem** kann vom Benutzer mit der Maus ein Element des gewünschten übergebenen Typs selektiert werden. Der Rückgabewert ist Null bei erfolgter Selektion und (-1) wenn an der Pickposition kein Element des gewünschten Typs gefunden wurde.

Siehe auch

Funktionen **ged_pickanyelem**, **ged_setpickelem**.

ged_setautocornrins - GED Modus für automatische Eckpunktgenerierung setzen (GED)

Synopsis

```
int ged_setautocornins(         // Status
      int [0,3];                // Eckpunkteingabemodus:
                                //    0 = Automatische Eckpunktgenerierung
deaktiviert
                                //    1 = Automatische Eckpunktgenerierung
                                //          für Leiterbahnen
                                //    2 = Automatische Eckpunktgenerierung
                                //          für Polygone
                                //    3 = Automatische Eckpunktgenerierung
                                //          für Leiterbahnen und Polygone
      );
```

Beschreibung

Die Funktion **ged_setautocornins** setzt den Eingabemodus für das wahlweise automatische Einfügen von Ecken bei der Generierung von Leiterbahnen und Polygonen. Der Eckpunkteingabemodus wird im **Layouteditor** üblicherweise über eine der Optionen Winkel+Raster oktagonal bzw. Nur Winkel oktagonal der Funktion Eingaberaster selektiert. Es wird ein Wert ungleich Null zurückgegeben, wenn ein ungültiger Oktagonalmodus spezifiziert wurde.

ged_setdblpar - GED Doubleparameter setzen (GED)

Synopsis

```
int ged_setdblpar(            // Status
      int [0,[;               // Parametertyp/-nummer:
                              //    0 = X-Koordinate letzte
                              //          Gruppenplatzierung (STD2)
                              //    1 = Y-Koordinate letzte
                              //          Gruppenplatzierung (STD2)
                              //    2 = Standardbauteilplatzierungswinkel
                              //    3 = Flächenautomatik Isolationsabstand (STD2)
                              //    4 = Flächenautomatik Min. Strukturgröße (STD2)
                              //    5 = Flächenautomatik Wärmefallenbreite (STD2)
                              //    6 = Flächenautomatik Wärmefallenisolation
                              //         (STD2)
                              //    7 = Flächenautomatik Schraffurlinienabstand
                              //         (STD2)
                              //    8 = Flächenautomatik Schraffurlinienbreite
                              //         (STD2)
                              //    9 = Flächenautomatik Schraffurlinienwinkel
                              //         (STD3)
                              //    10 = Netzsichtbarkeitsdialog
                              //          Netznamenskontollelementbreite
                              //    11 = Standardtextgröße (STD2)
                              //    12 = DRC-Abstandshaltung Textgröszlig;e (STD2)
                              //    13 = Autoplacement Bauteilexpansion (STD2)
                              //    14 = Autoplacement Bauteilpinfaktor [0, 1.0]
                              //    15 = Autoplacement Segmentpassung [0, 1.0]
                              //    16 = Autoplacement Bauteilkonturoffset (STD2)
                              //    17 = Standardtextplatzierungswinkel (STD3)
                              //    18 = Autorouter Umrandungsabstand (STD2)
                              //    19 = Autorouter Abstand Wärmefalle
                              //          zu Bohrung (STD2)
                              //    20 = Autorouter Abstand Isolation
                              //          zu Bohrung (STD2)
                              //    21 = Autorouter Maximale
                              //          Versorgungslänge (STD2)
                              //    22 = Autorouter Angefordertes Spezial
                              //          Routing-Raster (STD2)
                              //    23 = Autorouter Split Power Plane
                              //          Abstandshaltung (STD2)
                              //    24 = Autorouter BGA-Rastertoleranz (STD2)
                              //    25 = Autorouter Maximale
                              //          SMD-Fanoutlänge (STD2)
                              //    26 = Autorouter Pin-Via-Mindestabstand (STD2)
                              //    27 = CAM Gerber Standardlinienbreite (STD2)
                              //    28 = CAM Minimaler Abstand Wärmefalle
                              //          zu Bohrung (STD2)
                              //    29 = CAM Minimaler Abstand Isolation
                              //          zu Bohrung (STD2)
                              //    30 = CAM Toleranz Abstand Wärmefalle
                              //          zu Bohrung (STD2)
                              //    31 = CAM Toleranz Abstand Isolation
                              //          zu Bohrung (STD2)
                              //    32 = CAM Breite
                              //          Versorgungslagenumrandung (STD2)
                              //    33 = CAM Versorgungslagen-
                              //          Isolationsabstand (STD2)
                              //    34 = Busbahnbreite (STD2)
                              //    35 = Busbahnabstand (STD2)
      double;                 // Parameterwert
      );
```

Beschreibung

Die Funktion **ged_setdblpar** dient dazu, Systemparameter vom Typ **double** im **Layouteditor** zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **ged_setdblpar** gesetzten Systemparametern können mit der Funktion **ged_getdblpar** abgefragt werden.

Siehe auch

Funktionen **ged_getdblpar**, **ged_getintpar**, **ged_getstrpar**, **ged_setintpar**, **ged_setstrpar**.

ged_setdrcmarkmode - GED DRC Fehleranzeigemodus setzen (GED)

Synopsis

```
int ged_setdrcmarkmode(        // Status
      int [0,1];               // DRC-Fehleranzeigemodus:
                               //    0 = Fehlermarkeranzeige mit Fehlerfarbe
                               //    1 = Fehlermarkeranzeige mit Highlightfarbe
      );
```

Beschreibung

Die Funktion **ged_setdrcmarkmode** wsetzt den Anzeigmodus für DRC-Fehlermarker. Der Funktionsrückgabewert ist ungleich Null, wenn ein ungültiger Anzeigemodus spezifiziert wurde.

Siehe auch

Funktion **ged_getdrcmarkmode**.

ged_setintpar - GED Integerparameter setzen (GED)

Synopsis

```
int ged_setintpar(              // Status
      int [0,[;                 // Parametertyp/-nummer:
                                //    0 = Pickpunktanzeigemodus:
                                //        0 = keine Pickpunktanzeige
                                //        1 = Pickpunktanzeige
                                //        2 = Pickpunktübersichtsanzeige
                                //        3 = Pickpunkteditieranzeige
                                //    1 = automatischer DRC beim Laden:
                                //        0 = kein automatischer DRC
                                //        1 = automatischer DRC nach Abfrage
                                //        2 = automatischer DRC ohne Abfrage
                                //  [ 2 = bae_setcolor benutzen ]
                                //    3 = Infoanzeigeflag:
                                //        0 = keine automatische Infoanzeige
                                //        1 = automatische Infoanzeige
                                //    4 = Infoanzeigemodus:
                                //        0 = keine Infoanzeige
                                //        1 = komplette Infoanzeige
                                //        2 = nur Kupferinfoanzeige
                                //    5 = Winkeleditierrichtung
                                //    6 = Bauteilmakro-DRC:
                                //        0 = kompletter DRC
                                //        1 = Bauteilmakros als geprüft betrachten
                                //    7 = Rastereckenabfragemodus:
                                //        0 = keine Rastereckenabfrage
                                //        1 = komplette Rastereckanabfrage
                                //        2 = Rastereckenabfrage im aktuellen
                                //              Fenster
                                //        3 = Rastereckenabfrage im dynamisch
                                //              aktualisierten Fenster
                                //    8 = Minconaktualisierungsmodus
                                //    9 = DRC Polygontyp-Abschaltungsbits
                                //    10 = Warnmeldemodus:
                                //        Bit 0: SCM-Änderungs-Warnungen
                                //                unterdrücken
                                //        Bit 1: Warnungen Gruppenselektion
                                //                Füllflächenproblempolygone
                                //                unterdrücken
                                //        Bit 2: Variantenvergleichswarnungen
                                //                unterdrücken
                                //        Bit 3: Warnungen über Beendingung des
                                //                Autoroutermodus unterdrückt
```

```
                              //    11 = Lagenbenutzung Scanmodus
                              //    12 = Flächenpolygoneditiermodus:
                              //        0 = keine geschlossenen Linienzüge
                              //        1 = immer geschlossene Linienzüge
                              //        2 = Abfrage zum Schließen von Linienzügen
                              //    13 = DRC-Abstandsanzeigemuster
                              //    14 = Leiterbahneditiermodus:
                              //        0 = Eingaberasterfang
                              //        1 = Pin-/Segmentfang bei erster
                              //             Leiterbahnecke
                              //  [ 15 = Systemparameter - kein Schreibzugriff ]
                              //    16 = Bestätigungsabfragelimit für Netz löschen
                              //    17 = Plotvorschaumodus:
                              //        0 = keine Plotvorschau
                              //        1 = Plotterstiftbreite
                              //    18 = DRC-Distanzanzeigemodus:
                              //        0 = keine DRC-Distanzanzeige
                              //        1 = Leiterbahndistanzlinien
                              //        2 = Flächendistanzlinien
                              //        3 = Leiterbahndistanzmuster
                              //        4 = Flächendistanzmuster
                              //    19 = Textlagenspiegelungsmodus:
                              //        0 = keine Textlagenspiegelung
                              //        1 = Dokumentarlagenspiegelung
                              //        2 = Signal- und Dokumentarlagenspiegelung
                              //    20 = Standardbauteilspiegelungsmodus
                              //    21 = Autosave Intervall
                              //    22 = Bauteilluftlinienanzeigemodus:
                              //        0 = Keine Luftlinienanzeige
                              //        1 = Statische Luftlinienanzeige
                              //        2 = Dynamische Luftlinienanzeige
                              //    23 = Winkelfreigabeumschaltmodus:
                              //        0 = Umschaltung Pickseite standard
                              //        1 = Umschaltung Raster
                              //        2 = Umschaltung kürzere Seite
                              //        3 = Umschaltung Editierrichtung
                              //    24 = Copper fill heat trap mode
                              //        0 = Direktanschluss
                              //        1 = Pin- & Via-Wärmefallen
                              //        2 = Pin-Wärmefallen
                              //        3 = Via-Wärmefallen
                              //        |4 = Flag - keine benachbarten Pins
                              //        |8 = Flag - Wärmefallenleiterbahnen
                              //        |16 = Flag - nur unverbundene Lagen
                              //    25 = Flächenautomatik Leiterbahnmodus
                              //        0 = Runde Ecken
                              //        1 = Oktagonale Exken
                              //        2 = Oktagonale Kreise
                              //        3 = Oktagonale Ecken & Kreise
                              //    26 = Flächenautomatik Inselerkennung:
                              //        0 = Inseln beibehalten
                              //        1 = Inseln löschen
                              //        2 = Inseln selektieren
                              //    27 = Flächenautomatik Innenflächenmodus:
                              //        0 = Innenfüllflächen füllen
                              //        1 = Innenfüllflächen ausparen
                              //        |2 = Ausparflächen mit Abstand
                              //    28 = Flächenautomatik
                              //          Wärmefallenverbingsanzahl
                              //    29 = Flächenautomatik Spitzwinkelmodus:
                              //        0 = Spitze Winkel flach
                              //        1 = Spitze Winkel rund
                              //    30 = Flächenautomatik Schraffurmodus:
                              //        0 = Linienschraffur
                              //        1 = Gitterschraffur
                              //        |2 = Editierbare Pfade erzeugen
```

```
//    31 = Netzsichtbarkeitsdialogmodus:
//        0 = Einzelspalte für
//             Netznamenslistenanzeige
//        1 = Mehrspaltige Netznamenslistenanzeige
//    32 = Gruppenbewegtdarstellung:
//        0 = Bewegtbild aus
//        1 = Nur Baugruppenlage
//        2 = Bewegtbild ein
//        3 = Bewegtbild alles
//    33 = Leiterbahngruppenselektionsmodus:
//        0 = Leiterbahn- & Viaselektion
//        1 = Leiterbahnselektion
//        2 = Viaselektion
//    34 = Vorzugslagenauswahl (LAY1) ohne Aktionen
//    35 = Zwischenablagetext-
//           Platzierungsanforderung
//    36 = Editierrichtung
//    37 = Mincon-Flächenmodus (Bitmuster):
//        0 = Kein Flächen-Mincon
//        |1 = Kupferflächen-Mincon
//        |2 = Potentialflächen-Mincon
//    38 = Gruppenwinkelfreigabemodus:
//        0 = Gruppenwinkel einhalten
//        1 = Gruppenwinkel automatisch freigeben
//    39 = Autoplacement Optimierungszahl
//    40 = Autoplacement Bauteiltausch
//    41 = Autoplacement Pin-/Gattertausch
//    42 = Autoplacement Spiegelungsmodus:
//        0 = Kein SMD-Spiegeln
//        1 = SMD-Spiegeln
//        2 = 2 Pin-SMD-Spiegeln
//        3 = Immer SMD-Spiegeln
//    43 = Autoplacement Rotationsmodus:
//        0 = 0-90 Grad Drehung
//        1 = 0-270 Grad Drehung
//        2 = 0 Grad Drehung
//        3 = 90 Grad Drehung
//        4 = 0 XOR 90 Grad Drehung
//    44 = Autoplacement Retrydurchgänge
//    45 = Autoplacement SMD-Rotationsmodus:
//        0 = 0-90 Grad Drehung
//        1 = 0-270 Grad Drehung
//        2 = 0 Grad Drehung
//        3 = 90 Grad Drehung
//        4 = 0 XOR 90 Grad Drehung
//    46 = Autoplacement Bauteilkonturlage (LAY1)
//    47 = Gruppenbearbeitung Elementfang:
//        0 = Alle Elemente selektieren
//        1 = Nur sichtbare Elemente selektieren
//    48 = Standardtextspiegelung und
//           Standardtextmodus (STD14|LAY14)
//    49 = Autorouter Anzahl Optimierungen
//    50 = Autorouter Optimierer Richtungsmodus
//    51 = Autorouter Max. Via-Zahl
//    52 = Autorouter Via-Kosten
//    53 = Autorouter Pinkanal-Kosten
//    54 = Autorouter Anti-Vorzugs-Richtungs-Kosten
//    55 = Autorouter Richtungsänderungs-Kosten
//    56 = Autorouter Packungs-Kosten
//    57 = Autorouter Dynamische-Dichtestatistik-
//           Basis
//    58 = Autorouter Max. Bäume pro Rip-Up
//    59 = Autorouter Max. Rip-Up Tiefe
//    60 = Autorouter Max. Rip-Up Wiederholungen
//    61 = Autorouter Via-Raster Index
//    62 = Autorouter Bus-Abknickkosten
//    63 = Autorouter Abstand-1-Kosten
//    64 = Autorouter Abstand-2-Kosten
//    65 = Autorouter Rip-Up-Kosten
//    66 = Autorouter Router Cleanup
//    67 = Autorouter Optimierer Cleanup
```

```
//    68 = Autorouter Versorgungsanschluss
//          Vektorunroutes
//    69 = Autorouter Zwischenspeichern
//    70 = Autorouter Verbindungseckenausgabemodus
//    71 = Autorouter Unroutesausgabesortiermodus
//    72 = Autorouter Ecken Spitzwinkel Modus
//    73 = Autorouter Richtungsmodus
//          existierende Bahnen
//    74 = Autorouter Standardverbindungs-Lagen-
//          Kosten
//    75 = Autorouter Busverbindungs-Lagen-Kosten
//    76 = Autorouter Offset Wellenausbreitung Limit
//    77 = Autorouter Via Raster Modus
//    78 = Autorouter Eingabedaten Fehlerprüfmodus
//    79 = Autorouter Pin Anschluss Modus
//    80 = Autorouter Halbraster-Modus
//    81 = Autorouter Off-Grid-Routing-Kosten
//    82 = Autorouter Buserkennung/Busrouting Modus
//    83 = Autorouter SMD Via-Vorlegen
//    84 = Autorouter Router Pin/Gate-Swap
//    85 = Autorouter Gridless Routing Modus
//    86 = Autorouter Inkrementaler Ausgabemodus
//    87 = Autorouter Vorzugsraster
//    88 = Autorouter Anti-Vorzugsraster-Kosten
//    89 = Autorouter Anti-Netzbereichs-Kosten
//    90 = Autorouter Letzter Optimiererparameter
//          Wechselmodus
//    91 = Autorouter Auto-Rip-Up Parameter Modus
//    92 = Autorouter Vorzugsrichtungsmodus
//    93 = Autorouter Optimierer Reihenfolge Modus
//    94 = Autorouter Via Rip-Up Flag
//    95 = Autorouter Routingfenster Randausdehnung
//    96 = Autorouter BGA Fan out
//    97 = Autorouter Pinentry Korrektur
//    98 = Autorouter Alternativer Viaversatz Modus
//    99 = Autorouter Voll-Viaauswertemodus
//    100 = Autorouter Micro Via Modus
//    101 = Autorouter Richtungsvorgabe
//           Maximalabweichung
//    102 = Autorouter Routingfenster Flag
//    103 = Autorouter Padentry Subgrid
//    104 = Autorouter Versorgungslagen Vias
//    105 = Autorouter Via Prüfmodus
//    106 = Autorouter Verbindungsanzahl große Netze
//    107 = Autorouter Autorouting aktiv Flag
//    108 = CAM Wärmefallenbasiswinkel
//    109 = Flag - Einzeleckenbearbeitung
//    110 = Flag - Größenänderung mit Eckenabrundung
//    111 = Flag - Anzeige ausgeblendeter DRC-Fehler
//    112 = Flag - DRC-Verletzung Elementscan
//    113 = Bauteilplatzierung -
//           Bahnendenbewegungsmodus
//        0 = Keine Bahnendenbewegung
//        1 = Bahnenden mitbewegen
//        2 = Bahnsegmente mitbewegen
//    114 = Flag - Polygonbearbeitung
//           Autocomplete-Modus
//    115 = Platinenumrandung alternative
//           Dokumentarlage (LAY1)
//    116 = Leiterbahnzusammenfassungsmodus:
//        0 = Bahnen nicht zusammenfassen
//        1 = Bahnen zusammenfassen
//        2 = Bahnzusammenfassungsabfrage
//    117 = Anzeigemodus Leiterbahnen (LAY15)
//    118 = Anzeigemodus Texte (LAY15)
//    119 = Anzeigemodus Kupferpolygone (LAY15)
//    120 = Anzeigemodus Sperrflächen (LAY15)
//    121 = Anzeigemodus Umrandungspolygon (LAY15)
//    122 = Anzeigemodus Potentialflächen (LAY15)
//    123 = Anzeigemodus Dokumentarlinien (LAY15)
//    124 = Anzeigemodus Dokumentarflächen (LAY15)
```

```
                        //    125 = Anzeigemodus Füllflächen mit
                        //           Ausschnittspolygonen (LAY15)
                        //    126 = Anzeigemodus Schraffurflächen (LAY15)
                        //    127 = Anzeigemodus Geteilte
                        //           Potentialflächen (LAY15)
                        //    128 = Flag - Farbtabelle gesichert
                        //    129 = Airlinefarbmodus:
                        //        0 = Airlinefarbe benutzen
                        //        1 = Lagenfarbe benutzen
                        //    130 = Airlineclippingmodus:
                        //        0 = Kein Airlineclipping
                        //        1 = Airlines ohne Zielpunkt im
                        //             Arbeitsbereich ausblenden
                        //    131 = Bahnkollisionsmodus:
                        //        -1 = Operationsabfrage
                        //        0 = Kollisionen ignorieren
                        //        1 = Kollidierende Bahnen löschen
                        //        2 = Kollidierende Segmente löschen
                        //        3 = Kollidierende Segmente abschneiden
                        //    132 = Abfragemodus für Layoutbahnverbindungen:
                        //        0 = Layoutbahnen nie verbinden
                        //        1 = Layoutbahnen immer verbinden
                        //        2 = Verbindungsmodus abfragen
                        //    133 = Abfragemodus für
                        //           Bauteilbahnverbindungen:
                        //        0 = Bauteilbahnen nie verbinden
                        //        1 = Bauteilbahnen immer verbinden
                        //        2 = Verbindungsmodus abfragen
                        //    135 = Polygonanzeigemodus beim Bewegen
                        //           von Elementen:
                        //        0 = Umrandungsanzeige
                        //        1 = Füllanzeige
                        //    136 = Airlineanzeigemodus beim Bewegen
                        //           von Gruppen:
                        //        0 = Airlineanzeige aus
                        //        1 = Airlines für Gruppenbauteilpins
                        //             anzeigen
                        //    137 = Abstandsprüfmodus für Bahnkollisionen:
                        //        0 = DRC-Abstand für Kollisionscheck
                        //             benutzen
                        //        1 = Nur Kreuzungen als Kollisionen
                        //             interpretieren
                        //    138 = Pickmodus für Leiterbahnbündel:
                        //        0 = Kontinuierlicher Segmentpick
                        //        1 = Pick erstes und letztes Bündelsegment
                        //    139 = Einfüge-Pickmodus für
                        //           Leiterbahnsegmente:
                        //        0 = 3-Klick Selektion
                        //        1 = 2-Klick Selektion
                        //    140 = Bauteil-DRC:
                        //        0 = kein Bauteil-Online-DRC
                        //        1 = Bauteil-Online-DRC
                        //    141 = Flag - Optimierung Bohrwerkzeugtabelle
                        //    142 = Busbahnanzahl
                        //    143 = Busbahnanzahl editieren
                        //    144 = Generierungsmodus Busbahnen:
                        //        0 = Bahnbündel generieren
                        //        1 = Separate Bahnen generieren
                        //    145 = Busbahn-Eckenmodus:
                        //        0 = Gewinkelte Ecken generieren
                        //        1 = Abgerundete Ecken gnerieren
                        //    146 = Siebdrucklage (LAY1)
                        //    147 = Anzeigmodus Makroumrandungen:
                        //        0 = Keine Makroumrandungsanzeige
                        //        1 = Makroumrandungen für bewegte
                        //             Referenzen anzeigen
                        //        2 = Makroumrandungen anzeigen
       int;             // Parameterwert
       );
```

Beschreibung

Die Funktion **ged_setintpar** dient dazu, Systemparameter vom Typ im **int** im **Layouteditor** zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **ged_setintpar** gesetzten Systemparametern können mit der Funktion **ged_getintpar** abgefragt werden.

Siehe auch

Funktionen **ged_getdblpar**, **ged_getintpar**, **ged_getstrpar**, **ged_setdblpar**, **ged_setstrpar**.

ged_setlaydefmode - GED Defaultlagenmodus setzen (GED)

Synopsis

```
int ged_setlaydefmode(          // Status
      int [0,2];                // Defaultlagenmodus:
                                //    0 = automatische Lagendefaultzuweisung
                                //         deaktivieren
                                //    1 = benutzte Eingabelage ist Lagendefault
                                //    2 = zuletzt benutzte Lage ist Lagendefault
      );
```

Beschreibung

Die Funktion **ged_setlaydefmode** setzt den Defaultlagenmodus im **Layouteditor**. Der Funktionsrückgabewert ist Null bei erfolgreicher Zuweisung und ungleich Null im Fehlerfall.

Siehe auch

Funktionen **ged_getlaydefmode**, **ged_getlayerdefault**, **ged_setlayerdefault**.

ged_setlayerdefault - GED Defaultlage setzen (GED)

Synopsis

```
int ged_setlayerdefault(        // Status
      int;                      // Lage (LAY1)
      );
```

Beschreibung

Die Funktion **ged_setlayerdefault** setzt die Defaultlage im **Layouteditor**. Der Funktionsrückgabewert ist Null bei erfolgreicher Zuweisung und ungleich Null im Fehlerfall.

Siehe auch

Funktionen **ged_getlaydefmode**, **ged_getlayerdefault**, **ged_setlaydefmode**.

ged_setmincon - GED Mincon-Funktion setzen (GED)

Synopsis

```
int ged_setmincon(              // Status
      int [0,8];                // Mincon Funktionstyp (LAY10)
      );
```

Beschreibung

Die Funktion **ged_setmincon** setzt den Mincon-Modus für die Airlineanzeige im **Layouteditor**. Es wird ein Wert ungleich Null zurückgegeben, wenn ein ungültiger Modus angegeben wurde.

ged_setnetattrib - GED Netzattribut setzen (GED)

Synopsis

```
int ged_setnetattrib(          // Status
      string;                  // Netzname
      string;                  // Attributname
      string;                  // Attributwert
      );
```

Beschreibung

Die Funktion **ged_setnetattrib** setzt für das namentlich spezifizierte Signalnetz den Wert des angegebenen Attributs. Die maximal speicherbare Stringlänge für den Attributwert beträgt 40 Zeichen. Der Rückgabewert ist Null bei erfolgreicher Attributwertdefinition, (-1) wenn kein gültiges Element geladen ist, (-2) bei fehlenden bzw. ungültigen Parametern, (-3) wenn das Netz nicht gefunden wurde, oder (-4) wenn das Attribut mit dem angegebenen Namen nicht am Netz definiert ist.

ged_setpathwidth - GED Bahnenstandardbreiten setzen (GED)

Synopsis

```
int ged_setpathwidth(          // Status
      double ]0.0,[;           // Schmal Standardbreite (STD2)
      double ]0.0,[;           // Breit Standardbreite (STD2)
      );
```

Beschreibung

Die Funktion **ged_setpathwidth** setzt die Standardbreiten für schmale und breite Leiterbahnen im **Layouteditor**. Es wird ein Wert ungleich Null zurückgegeben, wenn ungültige Breiten spezifiziert wurden.

ged_setpickelem - Set GED default pick element (GED)

Synopsis

```
int ged_setpickelem(           // Status
      index L_FIGURE;          // Defaultpickelement
      );
```

Beschreibung

Die Funktion **ged_setpickelem** setzt ein Defaultelement für nachfogende Pickoperationen im **Layouteditor**. Der Funktionsrückgabewert ist ungleich Null im Fehlerfall.

Siehe auch

Funktion **ged_pickelem**.

ged_setpickmode - GED Elementpickmodus setzen (GED)

Synopsis

```
int ged_setpickmode(           // Status
      int [0,2];               // Elementpickmodus:
                               //    0 = Pick über Vorzugslage
                               //    1 = Pick mit Elementauswahl
                               //    2 = Pick exklusiv über Vorzugslage
      );
```

Beschreibung

Die Funktion **ged_setpickmode** setzt den Elementpickmodus im **Layouteditor**. Der Funktionsrückgabewert ist ungleich Null bei Spezifikation eines ungültigen Elementpickmodus.

Siehe auch

Funktion **ged_getpickmode**.

ged_setpickpreflay - GED Vorzugslage setzen (GED)

Synopsis

```
int ged_setpickpreflay(        // Status
      int;                     // Lage (LAY1)
      );
```

Beschreibung

Die Funktion **ged_setpickpreflay** setzt die Vorzugslage für Elementwahl im **Layouteditor**. Es wird ein Wert ungleich Null zurückgegeben, wenn keine gültige Lage angegeben wurde.

ged_setplantoplay - GED oberste Lage setzen (GED)

Synopsis

```
int ged_setplantoplay(         // Status
      int [0,99];              // Lage (LAY1)
      );
```

Beschreibung

Die Funktion **ged_setplantoplay** setzt die oberste Lage im **Layouteditor**. Es wird ein Wert ungleich Null zurückgegeben, wenn keine gültige Signallage angegeben wurde.

ged_setsegmovmode - GED Leiterbahnsegmentbewegungsmodus setzen (GED)

Synopsis

```
int ged_setsegmovmode(         // Status
      int [0,12];              // Leiterbahnsegmentbewegungsmodus:
                               //    0 = Mit Nachbarn bewegen
                               //    1 = Ohne Nachbarn bewegen
                               //    2 = Nachbarn anpassen
                               //    3 = Nachbarn ohne Durchkontaktierungen
                               //         anpassen
                               //    8 = Nur unmittelbare Nachbarn anpassen
                               //    |4 = Offene Leiterbahnenden folgen
                               //          Segmentbewegung
      );
```

Beschreibung

Die Funktion **ged_setsegmovmode** setzt den Leiterbahnsegmentbewegungsmodus im **Layouteditor**. Es wird ein Wert ungleich Null zurückgegeben, wenn ein ungültiger Modus angegeben wurde.

Siehe auch

Funktion **ged_getsegmovmode**.

ged_setstrpar - GED Stringparameter setzen (GED)

Synopsis

```
int ged_setstrpar(              // Status
      int [0,[;                 // Parametertyp/-nummer:
                                //  [ 0 = Systemparameter - kein Schreibzugriff ]
                                //  [ 1 = Systemparameter - kein Schreibzugriff ]
                                //    2 = Zuletzt platzierter Textstring
                                //    3 = Standardbibliotheksname
                                //  [ 4 = Systemparameter - kein Schreibzugriff ]
                                //    5 = Bohrbauteil-/Bohrpinnamensbasis
                                //    6 = Bohrbauteilmakronamensmuster
                                //    7 = Bohrpinmakronamensmuster
                                //    8 = Eingabeaufforderung Ersatzstring
                                //  [ 9 = Systemparameter - kein Schreibzugriff ]
                                //    10 = Autosave path name
      string;                   // Parameterwert
      );
```

Beschreibung

Die Funktion **ged_setstrpar** dient dazu, Systemparameter vom Typ im `string` im **Schaltplaneditor** zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **ged_setstrpar** gesetzten Systemparametern können mit der Funktion **ged_getstrpar** abgefragt werden.

Siehe auch

Funktionen **ged_getdblpar**, **ged_getintpar**, **ged_getstrpar**, **ged_setdblpar**, **ged_setintpar**.

ged_setviaoptmode - GED Leiterbahnviaoptimierungsmodus setzen (GED)

Synopsis

```
int ged_setviaoptmode(          // Status
      int [0,1];                // Leiterbahnviaoptimierungsmodus:
                                //    0 = Viaoptimierung
                                //    1 = Vias beibehalten
      );
```

Beschreibung

Die Funktion **ged_setviaoptmode** setzt den Leiterbahnviaoptimierungsmodus im **Layouteditor**. Es wird ein Wert ungleich Null zurückgegeben, wenn ein ungültiger Modus angegeben wurde.

Siehe auch

Funktion **ged_getviaoptmode**.

ged_setwidedraw - GED Breitendarstellung setzen (GED)

Synopsis

```
int ged_setwidedraw(            // Status
      double ]0.0,[;            // Breite (STD2)
      );
```

Beschreibung

Die Funktion **ged_setwidedraw** setzt die Breite, ab der im **Layouteditor** Leiterbahnen auf dem Bildschirm in Flächendarstellung angezeigt werden. Es wird ein Wert ungleich Null zurückgegeben, wenn eine ungültige Breite spezifiziert wurde.

ged_storedrill - GED Bohrung platzieren (GED)

Synopsis

```
int ged_storedrill(            // Status
      double;                  // X-Koordinate (STD2)
      double;                  // Y-Koordinate (STD2)
      double ]0.0,[;           // Radius (STD2)
      int [0,[;                // Bohrklasse (LAY5)
      );
```

Beschreibung

Die Funktion **ged_storedrill** platziert eine Bohrung mit den angegebenen Parametern auf dem gegenwärtig geladenen Layoutelement. Der Rückgabewert ist ungleich Null, wenn ungültige Daten übergeben wurden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in **forall**-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ged_storepart - GED Bauteil platzieren (GED)

Synopsis

```
int ged_storepart(             // Status
      string;                  // Bauteilname
      string;                  // Bauteil Bibiotheksteilname
      double;                  // X-Koordinate (STD2)
      double;                  // Y-Koordinate (STD2)
      double;                  // Drehwinkel (STD3)
      int [0,1];               // Spiegelungsmodus (STD14)
      );
```

Beschreibung

Die Funktion **ged_storepart** platziert ein Bauteil mit den angegebenen Parametern auf dem gegenwärtig geladenen Layoutelement. Wird eine Leerzeichenkette für den Bauteilnamen übergeben, so wird das nächste unplatzierte Bauteil der Netzliste verwendet. Der Rückgabewert ist Null, wenn das Bauteil erfolgreich platziert wurde, (1) wenn die Bauteilpins nicht mit der Netzliste übereinstimmen, (-1) bei ungültigen Daten, (-2) wenn alle Bauteile bereits platziert sind, (-3) wenn das Bauteil schon platziert ist, (-4) wenn es nicht ladbar ist und (-6) wenn die Bauteildaten nicht in die Jobdatenbank kopiert werden konnten.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in **forall**-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ged_storepath - GED Bahn platzieren (GED)

Synopsis

```
int ged_storepath(              // Status
      int [0,99];               // Leiterbahnlage (LAY1)
      double ]0.0,[;            // Leiterbahnbreite (STD2)
      );
```

Beschreibung

Die Funktion **ged_storepath** erzeugt aus der internen Punktliste unter Verwendung der angegebenen Parameter eine Leiterbahn auf dem aktuell geladenen Layout- bzw. Bauteil. Der Rückgabewert ist gleich Null, wenn die Bahn erfolgreich platziert wurde, (-1) wenn kein gültiges Element geladen ist, (-2) bei ungültigen Parametern, oder (-3) wenn die Punktliste ungültig ist.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

Siehe auch

Funktionen **bae_storepoint**, **ged_drcpath**.

ged_storepoly - GED Fläche platzieren (GED)

Synopsis

```
int ged_storepoly(              // Status
      int;                      // Polygonlage (LAY1)
      int [1,9];                // Polygontyp (LAY4)
      string;                   // Polygonnetzname (für Pot.- und Füllbereich)
      int [0,18];               // Polygonspiegelungsmodus (LAY3)
      );
```

Beschreibung

Die Funktion **ged_storepoly** generiert aus der mit **bae_storepoint** erzeugten internen Punktliste unter Verwendung der angegebenen Parameter eine Fläche auf dem gegenwärtig geladenen Layoutelement. Der Rückgabewert ist gleich Null, wenn die Fläche erfolgreich platziert wurde, (-1) wenn kein gültiges Element geladen ist, (-2) bei ungültigen Parametern, oder (-3) wenn die Punktliste für den gegebenen Flächentyp ungültig ist.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

Siehe auch

Funktionen **bae_storepoint**, **ged_drcpoly**.

ged_storetext - GED Text platzieren (GED)

Synopsis

```
int ged_storetext(              // Status
      string;                   // Textzeichenkette
      double;                   // Text-X-Koordinate (STD2)
      double;                   // Text-Y-Koordinate (STD2)
      double;                   // Textdrehwinkel (STD3)
      double ]0.0,[;            // Textgröße (STD2)
      int;                      // Textlage (LAY1)
      int;                      // Textspiegelungsmodus und Textstil (STD14|LAY14)
      );
```

Beschreibung

Die Funktion **ged_storetext** platziert einen Text mit den angegebenen Parametern auf dem aktuell geladenen Layoutelement. Der Rückgabewert ist ungleich Null, wenn ungültige Daten übergeben wurden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden. Es können maximal 40 Zeichen der übergebenen Textzeichenkette gespeichert werden. Bei Übergabe längerer Zeichenketten gibt die Funktionen den Fehlerstatus zur Kennzeichnung ungültiger Parameter zurück.

Siehe auch

Funktion **ged_attachtextpos**.

ged_storeuref - GED Via bzw. Pad platzieren (GED)

Synopsis

```
int ged_storeuref(              // Status
      string;                   // Referenz Bibiotheksteilname
      double;                   // X-Koordinate (STD2)
      double;                   // Y-Koordinate (STD2)
      double;                   // Drehwinkel (STD3)
      int;                      // Lagenoffset (für Pad auf Padstack)
      int [0,1];                // Spiegelung (STD14) (für Pad auf Padst.)
      );
```

Beschreibung

Die Funktion **ged_storeuref** platziert eine namenlose Referenz mit den angegebenen Parametern auf dem gegenwärtig geladenen Layoutelement. Namenlose Referenzen sind die Vias auf Layout- bzw. Bauteilebene und die Pads auf Padstackebene. Spiegelung, Lagenoffset und Drehwinkel werden für Vias ignoriert. Der Rückgabewert ist gleich Null, wenn die Referenz erfolgreich platziert wurde, (-1) bei ungültigen Daten, (-2) wenn sie nicht ladbar ist und (-3) wenn die Referenzdaten nicht in die Jobdatenbank kopiert werden konnten.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

Siehe auch
Funktion **ged_drcpoly**.

C.4.3 Autorouter-Funktionen

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp AR zugeordnet, d.h. diese Funktionen können im **Autorouter** aufgerufen werden:

ar_asklayer - Autorouter Lagenauswahl (AR)

Synopsis

```
int ar_asklayer(                // Status
      & int;                    // Lagenrückgabe (LAY1|LAY9)
      int [0,5];                // Lagenabfragetyp:
                                //    0 = Dokumentarlagen und Signallagen
                                //    1 = Signallagen
                                //    2 = Signallagen
                                //        (inklusive Oberste Lage und Alle Lagen)
                                //    3 = Dokumentarlagen
                                //    4 = Signallagen und Versorgungslagen
                                //    5 = beliebige Anzeigeelementtypen
      );
```

Beschreibung

Die Funktion **ar_asklayer** aktiviert im **Autorouter** ein Lagenauswahlmenü. Der Lagenabfragetyp gibt an, welche Lagen bzw. Anzeigelementtypen zur Auswahl angeboten werden. Der Rückgabewert ist Null bei erfolgter Lagenwahl oder (-1) bei Wahl des Menüpunktes Abbruch.

ar_delelem - Autorouter Element löschen (AR)

Synopsis

```
int ar_delelem(                 // Status
      & index L_FIGURE;         // Element
      );
```

Beschreibung

Die Funktion **ar_delelem** löscht das übergebene Element aus der Elementliste. Der Rückgabewert ist Null bei erfolgter Löschung und (-1), wenn das übergebene Element ungültig ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

Siehe auch

Funktion **ar_drawelem**.

ar_drawelem - Autorouter Elementanzeige aktualisieren (AR)

Synopsis

```
void ar_drawelem(
      index L_FIGURE;           // Element
      int [0, 4];               // Zeichenmodus (STD19)
      );
```

Beschreibung

Die Funktion **ar_drawelem** aktualisiert die Anzeige des angegebenen Elements unter Verwendung des spezifizierten Zeichenmodus.

Siehe auch

Funktion **ar_delelem**.

ar_elemangchg - Autorouter Elementwinkel ändern (AR)

Synopsis

```
int ar_elemangchg(              // Status
      & index L_FIGURE;         // Element
      double;                   // Neuer Winkel (STD3)
      );
```

Beschreibung

Die Funktion **ar_elemangchg** ändert den Drehwinkel des übergebenen Elements. Der Drehwinkel wird ausgehend vom Nullwinkel eingestellt, d.h. der vorhergehende Drehwinkel des Elements hat keinen Einfluss auf das Ergebnis. Die Winkelangabe wird als Bogenmaßwert interpretiert. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht drehbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in **forall**-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ar_elemfixchg - Autorouter Element fixiert-Flag ändern (AR)

Synopsis

```
int ar_elemfixchg(              // Status
      & index L_FIGURE;         // Element
      int [0,1];                // Neues fixiert Flag (STD11)
      );
```

Beschreibung

Die Funktion **ar_elemfixchg** ändert den Fixiert-Modus des übergebenen Elements. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht fixierbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in **forall**-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ar_elemlaychg - Autorouter Elementlage ändern (AR)

Synopsis

```
int ar_elemlaychg(              // Status
      & index L_FIGURE;         // Element
      int;                      // Neue Lage (LAY1)
      );
```

Beschreibung

Die Funktion **ar_elemlaychg** ändert die Lagenzugehörigkeit des übergebenen Elements. Bei Bohrungen gibt die Lage die Bohrungsklasse an. Die Lage kann für Flächen, Leiterbahnen, Bohrungen, Texte und Pads auf Padstackebene geändert werden. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn die Lage nicht änderbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in **forall**-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ar_elemmirrchg - Autorouter Elementspiegelung ändern (AR)

Synopsis

```
int ar_elemmirrchg(             // Status
      & index L_FIGURE;         // Element
      int [0,18];               // Neuer Spiegelungsmodus (STD14|LAY3)
      );
```

Beschreibung

Die Funktion **ar_elemmirrchg** ändert den Spiegelungsmodus des übergebenen Elements. Der Spiegelungsmodus kann bei Flächen, Texten und Referenzen geändert werden. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig ist oder (-2) wenn es keinen Spiegelungsmodus besitzt. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ar_elemposchg - Autorouter Elementposition ändern (AR)

Synopsis

```
int ar_elemposchg(              // Status
      & index L_FIGURE;         // Element
      double;                   // X-Position (STD2)
      double;                   // Y-Position (STD2)
      );
```

Beschreibung

Die Funktion **ar_elemposchg** ändert die Position des übergebenen Elements. Bei Flächen/Leiterbahnen wird die Fläche/Leiterbahn so verschoben, dass der erste Punkt der Fläche/Leiterbahn auf der angegebenen Position zu liegen kommt. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht positionierbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ar_elemsizechg - Autorouter Elementgröße ändern (AR)

Synopsis

```
int ar_elemsizechg(             // Status
      & index L_FIGURE;         // Element
      double;                   // Neue Größe (STD2)
      );
```

Beschreibung

Die Funktion **ar_elemsizechg** ändert die Größe des übergebenen Elements. Bei Leiterbahnen wird mit der Größe die Leiterbahnbreite spezifiziert. Eine Größenänderung ist nur bei Texten, Bohrungen und Leiterbahnen möglich. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht größenveränderbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ar_getdblpar - Autorouter Doubleparameter abfragen (AR)

Synopsis

```
int ar_getdblpar(              // Status
      int [0,[;                // Parametertyp/-nummer:
                               //    0 = Netzsichtbarkeitsdialog
Netznamenskontollelementbreite
                               //    1 = Benutzerdefiniertes Routingraster
      & double;                // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **ar_getdblpar** dient der Abfrage von mit **ar_setdblpar** im **Autorouter** gesetzten Parametern vom Typ `double`. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **ar_getintpar**, **ar_getstrpar**, **ar_setdblpar**, **ar_setintpar**, **ar_setstrpar**.

ar_getintpar - Autorouter Integerparameter abfragen (AR)

Synopsis

```
int ar_getintpar(              // Status
      int [0,[;                // Parametertyp/-nummer:
                               //    0 = Farbcode für Oberste Lage
                               //    1 = Minconaktualisierungsmodus
                               //    2 = Warnmeldemodus:
                               //        Bit 0: SCM-Änderungs-Warnungen unterdrückt
                               //        Bit 1: im Autorouter nicht benutzt
                               //        Bit 2: Variantenvergleichswarnungen
                               //                unterdrückt
                               //    3 = Autosave-Intervall
                               //    4 = Netzsichtbarkeitsdialogmodus:
                               //        0 = Einzelspalte für
                               //             Netznamenslistenanzeige
                               //        1 = Mehrspaltige Netznamenslistenanzeige
                               //    5 = Routingsignallagenanzahl
                               //    6 = Routingrastercode:
                               //        0 = 1/20 Zoll (1.27 mm) Standard
                               //        1 = 1/40 Zoll (0.635 mm) Standard
                               //        2 = 1/50 Zoll (0.508 mm) Standard
                               //        3 = 1/60 Zoll (0.4233 mm) Standard
                               //        4 = 1/80 Zoll (0.3175 mm) Standard
                               //        5 = 1/100 Zoll (0.254 mm) Standard
                               //        6 = 1/40 Zoll (0.635 mm) ohne Versatz
                               //        7 = 1/60 Zoll (0.4233 mm) ohne Versatz
                               //        8 = 1/80 Zoll (0.3175 mm) ohne Versatz
                               //        9 = 1/100 Zoll (0.254 mm) mit Versatz
                               //        -1 = Anderes Raster ohne Versatz
                               //        -2 = Anderes Raster mit Versatz
                               //    7 = Mincon-Flächenmodus (Bitmuster):
                               //        0 = Kein Flächen-Mincon
                               //        |1 = Kupferflächen-Mincon
                               //        |2 = Potentialflächen-Mincon
                               //    8 = Flag - Farbtabelle gesichert
                               //    9 = Airlinefarbmodus:
                               //        0 = Airlinefarbe benutzen
                               //        1 = Lagenfarbe benutzen
                               //    10 = Flag - Optimierung Bohrwerkzeugtabelle
      & int;                   // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **ar_getintpar** dient der Abfrage von mit **ar_setintpar** im **Autorouter** gesetzten Integerparametern. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **ar_getdblpar**, **ar_getstrpar**, **ar_setdblpar**, **ar_setintpar**, **ar_setstrpar**.

ar_getmincon - Autorouter Mincon-Funktion abfragen (AR)

Synopsis

```
int ar_getmincon(              // Mincon Funktionstyp (LAY10)
      );
```

Beschreibung

Der Rückgabewert der Funktion **ar_getmincon** entspricht dem im **Autorouter** aktuell eingestellten Wert des Mincon-Modus für die Airlineanzeige (LAY10).

ar_getpickpreflay - Autorouter Vorzugslage abfragen (AR)

Synopsis

```
int ar_getpickpreflay(          // Vorzugslage (LAY1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **ar_getpickpreflay** entspricht der Vorzugslage (LAY1) für Elementwahl im **Autorouter**.

ar_getstrpar - Autorouter Stringparameter abfragen (AR)

Synopsis

```
int ar_getstrpar(               // Returns status
      int [0,[;                 // Parameter type/number:
                                //    0 = Autosave path name
      & string;                 // Returns parameter value
      );
```

Beschreibung

Die Funktion **ar_getstrpar** dient der Abfrage von im **Autorouter** gesetzten Stringparametern. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **ar_getdblpar**, **ar_getintpar**, **ar_setdblpar**, **ar_setintpar**, **ar_setstrpar**.

ar_getwidedraw - Autorouter Breitendarstellung abfragen (AR)

Synopsis

```
double ar_getwidedraw(          // Breite (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **ar_getwidedraw** entspricht der Breite, ab der im **Autorouter** Leiterbahnen auf dem Bildschirm in Flächendarstellung angezeigt werden.

ar_highlnet - Autorouter Highlight Netz ein/aus (AR)

Synopsis

```
int ar_highlnet(                // Status
      int [0,[;                 // Netznummer
      int [0,1];                // Highlightmodus (0 = aus, 1 = ein)
      );
```

Beschreibung

Die Funktion **ar_highlnet** setzt den Highlightmodus des Netzes mit der übergebenen Netznummer. Es wird ein Wert ungleich Null zurückgegeben, wenn ein ungültiger Highlightmodus oder eine ungültige Netznummer angegeben wurde.

ar_partnamechg - Autorouter Bauteilname in Netzliste ändern (AR)

Synopsis

```
int ar_partnamechg(            // Status
      string;                  // Alter Bauteilname
      string;                  // Neuer Bauteilname
      );
```

Beschreibung

Die Funktion **ar_partnamechg** ändert den Namen des angegebenen Bauteils in der Netzliste. Ein Rückgabewert von Null zeigt eine erfolgreiche Änderung an. Bei ungültigen Eingabedaten wird (-1) zurückgegeben, (-2) wenn das Bauteil nicht platziert ist, (-3) wenn das Bauteil nicht in der Netzliste vorhanden ist, (-4) wenn der neue Name schon definiert ist und (-5) wenn versucht wurde, in einem Programmlauf ein Bauteil mehrfach umzubenennen (z.B. **a** in **b** und anschließend **b** in **c**). Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion verändert die Netzliste und erfordert daher einen anschließenden Backannotation-Lauf. Die Funktion sollte auch nicht innerhalb von **L_CPART**-Index-Schleifen aufgerufen werden, da die vor dem Aufruf der Funktion belegten **L_CPART**-Indexvariablen anschließend ungültig sind.

ar_pickelem - Autorouter Element mit Maus selektieren (AR)

Synopsis

```
int ar_pickelem(               // Status
      & index L_FIGURE;        // Rückgabe Element
      int [1,9];               // Elementtyp (LAY6 außer 7)
      );
```

Beschreibung

Mit der Funktion **ar_pickelem** kann vom Benutzer mit der Maus ein Element des gewünschten übergebenen Typs selektiert werden. Der Rückgabewert ist Null bei erfolgter Selektion und (-1) wenn an der Pickposition kein Element des gewünschten Typs gefunden wurde.

ar_setdblpar - Autorouter Doubleparameter setzen (AR)

Synopsis

```
int ar_setdblpar(              // Status
      int [0,[;                // Parametertyp/-nummer:
                               //    0 = Netzsichtbarkeitsdialog
                               //          Netznamenskontollelementbreite
                               //  [ 1 = Systemparameter - kein Schreibzugriff ]
      double;                  // Parameterwert
      );
```

Beschreibung

Die Funktion **ar_setdblpar** dient dazu, Systemparameter vom Typ `double` im **Autorouter** zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **ar_setdblpar** gesetzten Systemparametern können mit der Funktion **ar_getdblpar** abgefragt werden.

Siehe auch

Funktionen **ar_getdblpar**, **ar_getintpar**, **ar_getstrpar**, **ar_setintpar**, **ar_setstrpar**.

ar_setintpar - Autorouter Integerparameter setzen (AR)

Synopsis

```
int ar_setintpar(              // Status
      int [0,[;                // Parametertyp/-nummer:
                               //  [ 0 = bae_setcolor benutzen ]
                               //    1 = Minconaktualisierungsmodus
                               //    2 = Warnmeldemodus:
                               //        Bit 0: SCM-Änderungs-Warnungen
                               //                unterdrücken
                               //        Bit 1: im Autorouter nicht benutzt
                               //        Bit 2: Variantenvergleichswarnungen
                               //                unterdrücken
                               //    3 = Autosave-Intervall
                               //    4 = Netzsichtbarkeitsdialogmodus:
                               //        0 = Einzelspalte für
                               //             Netznamenslistenanzeige
                               //        1 = Mehrspaltige Netznamenslistenanzeige
                               //  [ 5 = Systemparameter - kein Schreibzugriff ]
                               //  [ 6 = Systemparameter - kein Schreibzugriff ]
                               //    7 = Mincon-Flächenmodus (Bitmuster):
                               //        0 = Kein Flächen-Mincon
                               //        |1 = Kupferflächen-Mincon
                               //        |2 = Potentialflächen-Mincon
                               //    8 = Flag - Farbtabelle gesichert
                               //    9 = Airlinefarbmodus:
                               //        0 = Airlinefarbe benutzen
                               //        1 = Lagenfarbe benutzen
                               //    10 = Flag - Optimierung Bohrwerkzeugtabelle
      int;                     // Parameterwert
      );
```

Beschreibung

Die Funktion **ar_setintpar** dient dazu, Systemparameter vom Typ im **int** im **Autorouter** zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **ar_setintpar** gesetzten Systemparametern können mit der Funktion **ar_getintpar** abgefragt werden.

Siehe auch

Funktionen **ar_getdblpar**, **ar_getintpar**, **ar_getstrpar**, **ar_setdblpar**, **ar_setstrpar**.

ar_setmincon - Autorouter Mincon-Funktion setzen (AR)

Synopsis

```
int ar_setmincon(              // Status
      int [0,8];               // Mincon Funktionstyp (LAY10)
      );
```

Beschreibung

Die Funktion **ar_setmincon** setzt den Mincon-Modus für die Airlineanzeige im **Autorouter**. Es wird ein Wert ungleich Null zurückgegeben, wenn ein ungültiger Modus angegeben wurde.

ar_setnetattrib - Autorouter Netzattribut setzen (AR)

Synopsis

```
int ar_setnetattrib(           // Status
      string;                  // Netzname
      string;                  // Attributname
      string;                  // Attributwert
      );
```

Beschreibung

Die Funktion **ar_setnetattrib** setzt für das namentlich spezifizierte Signalnetz den Wert des angegebenen Attributs. Die maximal speicherbare Stringlänge für den Attributwert beträgt 40 Zeichen. Der Rückgabewert ist Null bei erfolgreicher Attributwertdefinition, (-1) wenn kein gültiges Element geladen ist, (-2) bei fehlenden bzw. ungültigen Parametern, (-3) wenn das Netz nicht gefunden wurde, oder (-4) wenn das Attribut mit dem angegebenen Namen nicht am Netz definiert ist.

ar_setpickpreflay - Autorouter Vorzugslage setzen (AR)

Synopsis

```
int ar_setpickpreflay(         // Status
      int;                     // Lage (LAY1)
      );
```

Beschreibung

Die Funktion **ar_setpickpreflay** setzt die Vorzugslage für Elementwahl im **Autorouter**. Es wird ein Wert ungleich Null zurückgegeben, wenn keine gültige Lage angegeben wurde.

ar_setplantoplay - Autorouter oberste Lage setzen (AR)

Synopsis

```
int ar_setplantoplay(          // Status
      int [0,99];              // Lage (LAY1)
      );
```

Beschreibung

Die Funktion **ar_setplantoplay** setzt die oberste Lage im **Autorouter**. Es wird ein Wert ungleich Null zurückgegeben, wenn keine gültige Signallage angegeben wurde.

ar_setstrpar - Autorouter Stringparameter setzen (GED)

Synopsis

```
int ar_setstrpar(              // Returns status
      int [0,[;                // Parameter type/number:
                               //    0 = Autosave path name
      string;                  // Parameter value
      );
```

Beschreibung

Die Funktion **ar_setstrpar** dient dazu, Systemparameter vom Typ im `string` im **Autorouter** zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **ar_setstrpar** gesetzten Systemparametern können mit der Funktion **ar_getstrpar** abgefragt werden.

Siehe auch

Funktionen **ar_getdblpar**, **ar_getintpar**, **ar_getstrpar**, **ar_setdblpar**, **ar_setintpar**.

ar_setwidedraw - Autorouter Breitendarstellung setzen (AR)

Synopsis

```
int ar_setwidedraw(             // Status
      double ]0.0,[;            // Breite (STD2)
      );
```

Beschreibung

Die Funktion **ar_setwidedraw** setzt die Breite, ab der im **Autorouter** Leiterbahnen auf dem Bildschirm in Flächendarstellung angezeigt werden. Es wird ein Wert ungleich Null zurückgegeben, wenn eine ungültige Breite spezifiziert wurde.

ar_storepart - Autorouter Bauteil platzieren (AR)

Synopsis

```
int ar_storepart(               // Status
      string;                   // Bauteilname
      string;                   // Bauteil Bibiotheksteilname
      double;                   // X-Koordinate (STD2)
      double;                   // Y-Koordinate (STD2)
      double;                   // Drehwinkel (STD3)
      int [0,1];                // Spiegelungsmodus (STD14)
      );
```

Beschreibung

Die Funktion **ar_storepart** platziert ein Bauteil mit den angegebenen Parametern auf dem gegenwärtig geladenen Layoutelement. Wird eine Leerzeichenkette für den Bauteilnamen übergeben, so wird das nächste unplatzierte Bauteil der Netzliste verwendet. Der Rückgabewert ist gleich Null, wenn das Bauteil erfolgreich platziert wurde, (-1) bei ungültigen Daten, (-2) wenn alle Bauteile bereits platziert sind, (-3) wenn das Bauteil schon platziert ist, (-4) wenn es nicht ladbar ist, (-5) wenn die Bauteilpins nicht mit der Netzliste übereinstimmen und (-6) wenn die Bauteildaten nicht in die Jobdatenbank kopiert werden konnten.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ar_storepath - Autorouter Bahn platzieren (AR)

Synopsis

```
int ar_storepath(               // Status
      int [0,99];               // Lage (LAY1)
      double ]0.0,[;            // Bahnbreite (STD2)
      );
```

Beschreibung

Die Funktion **ar_storepath** erzeugt aus der internen Punktliste unter Verwendung der angegebenen Parameter eine Leiterbahn auf dem aktuell geladenen Layout bzw. Bauteil. Der Rückgabewert ist gleich Null, wenn die Bahn erfolgreich platziert wurde, (-1) wenn kein gültiges Element geladen ist, (-2) bei ungültigen Parametern, oder (-3) wenn die Punktliste ungültig ist.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ar_storeuref - Autorouter Via bzw. Pad platzieren (GED)

Synopsis

```
int ar_storeuref(             // Status
      string;                 // Referenz Bibiotheksteilname
      double;                 // X-Koordinate (STD2)
      double;                 // Y-Koordinate (STD2)
      double;                 // Drehwinkel (STD3)
      int;                    // Lagenoffset (für Pad auf Stack)
      int [0,1];              // Spiegelung (STD14) (für Pad auf Padst.)
      );
```

Beschreibung

Die Funktion **ar_storeuref** platziert eine namenlose Referenz mit den angegebenen Parametern auf dem gegenwärtig geladenen Layoutelement. Namenlose Referenzen sind die Vias auf Layout- bzw. Bauteilebene und die Pads auf Padstackebene. Spiegelung, Lagenoffset und Drehwinkel werden für Vias ignoriert. Der Rückgabewert ist gleich Null, wenn die Referenz erfolgreich platziert wurde, (-1) bei ungültigen Daten, (-2) wenn sie nicht ladbar ist und (-3) wenn die Referenzdaten nicht in die Jobdatenbank kopiert werden konnten.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **L_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

C.4.4 CAM-Prozessor-Funktionen

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp CAM zugeordnet, d.h. diese Funktionen können im **CAM-Prozessor** aufgerufen werden:

cam_askplotlayer - CAM Plotlagenauswahl (CAM)

Synopsis

```
int cam_askplotlayer(          // Status
      & int;                   // Lagenrückgabe (LAY1)
      );
```

Beschreibung

Die Funktion **cam_askplotlayer** aktiviert im **CAM-Prozessor** ein Menü zur Auswahl der Plot- bzw. Ausgabelage. Der Rückgabewert ist Null bei erfolgreicher Lagenauswahl oder (-1) bei Wahl des Menüpunktes Abbruch.

cam_getdblpar - CAM Doubleparameter abfragen (CV)

Synopsis

```
int cam_getdblpar(             // Status
      int [0,[;                // Parametertyp/-nummer:
                               //    [ 0 = Systemparameter, nur Schreibzugriff ]
                               //    1 = Bitmapplotpixelauflösung (STD2)
                               //    2 = Letztes Bitmapplotpixelverhätnis
      & double;                // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **cam_getdblpar** dient der Abfrage von mit **cam_setdblpar** in **CAM-Prozessor** gesetzten Parametern vom Typ **`double`**. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **cam_getintpar**, **cam_setdblpar**, **cam_setintpar**.

cam_getdrlaccuracy - CAM Bohrwerkzeugtoleranz abfragen (CAM)

Synopsis

```
double cam_getdrlaccuracy(     // Bohrwerkzeugtoleranz (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **cam_getdrlaccuracy** entspricht dem aktuell im **CAM-Prozessor** eingestellten Wert für die Toleranz des Bohrwerkzeugs.

Siehe auch
Funktion **cam_setdrlaccuracy**.

cam_getgenpltparam - CAM allgemeine Plotparameter abfragen (CAM)

Synopsis

```
void cam_getgenpltparam(
      & int;                 // Alle Lagen Modus:
                             //    0 = aus
                             //    1 = ein
      & int;                 // Umrandungsmodus:
                             //    0 = aus
                             //    1 = ein
      & int;                 // Drehung:
                             //    0 = Drehung 0 Grad
                             //    1 = Drehung 90 Grad links
      & int;                 // Spiegelung (CAM1)
      & int;                 // Passermarkenmodus:
                             //    0 = aus
                             //    1 = ein
      & double;              // Plotgenauigkeit (STD2)
      & double;              // Plot Nullpunkt X-Koordinate (STD2)
      & double;              // Plot Nullpunkt Y-Koordinate (STD2)
      );
```

Beschreibung

Die Funktion **cam_getgenpltparam** gibt die aktuell im **CAM-Prozessor** eingestellten allgemeinen Plotparameter zurück.

cam_getgerberapt - CAM Gerberblendendefinition abfragen (CAM)

Synopsis

```
int cam_getgerberapt(        // Status
      int [1,900];           // Blendentabellenindex
      & int;                 // Blende D-Code:
                             //    10..999 = gültige D-Codes
                             //       (-1) = Blende nicht definiert
      & int;                 // Blendentyp:
                             //    0 = Spezialblende
                             //    1 = Runde Blende
                             //    2 = Quadratische Blende
                             //    3 = Therm. Blende (Wärmefalle)
                             //    4 = Rechteckige Blende
      & int;                 // Blendenzeichenmodus:
                             //    0 = Blende für alle Zeichenmodi
                             //    1 = Blende zum Blitzen
                             //    2 = Blende für Linienstrukturen
      & double;              // Blendengröße (STD2)
      & double;              // Blendengröße 2 (STD2)
      );
```

Beschreibung

Die Funktion **cam_getgerberapt** ermittelt die Definition der unter dem angegebenen Index der aktuell im **CAM-Prozessor** geladenen Blendentabelle eingetragenen Gerberblende. Wird der Wert (-1) für den D-Code zurückgegeben, dann bedeutet dies, dass keine Blende an der angegebenen Tabellenposition definiert ist. Der Funktionsrückgabewert ist ungleich Null bei fehlenden bzw. ungültigen Parametern.

cam_getgerberparam - CAM Gerber-Parameter abfragen (CAM)

Synopsis

```
void cam_getgerberparam(
      & string;              // Gerber Dateiname
      & double;              // Standardlinienbreite (STD2)
      & int;                 // Gerber Format (CAM4)
      & int;                 // Gerber Optimierung:
                             //    0 = Optimierung aus
                             //    1 = Optimierung ein
      & int;                 // Füllverfahren:
                             //    0 = Linien-Füllen
                             //    1 = Multiblenden-Füllen
                             //    2 = G36/G36-Füllmodus
      & int;                 // Kreisbogen-Ausgabemodus:
                             //    0 = Kreisbogen-Interpolation
                             //    1 = Ausg. mit Gerber I/J-Befehlen
      & int;                 // Extended Gerber (RS-274-X) Modus:
                             //    0 = Kein Extended Gerber
                             //    1 = Extended Gerber mit
                             //          Standard-Blendentabelle
                             //    2 = Extended Gerber mit
                             //          dynamischer Blendentabelle
      );
```

Beschreibung

Die Funktion **cam_getgerberparam** gibt die aktuell im **CAM-Prozessor** eingestellten Gerber-Plotparameter zurück.

cam_gethpglparam - CAM HP-GL-Parameter abfragen (CAM)

Synopsis

```
void cam_gethpglparam(
      & string;              // HP-GL Plotdateiname
      & double;              // HP-GL Maßstab
      & double;              // HP-GL Geschwindigkeit (-1.0 volle Geschw.)
      & double;              // HP-GL Stiftbreite (STD2)
      & int;                 // HP-GL Füllmodus:
                             //    0 = Füllen aus
                             //    1 = Füllen ein
      );
```

Beschreibung

Die Funktion **cam_gethpglparam** gibt die aktuell im **CAM-Prozessor** eingestellten HP-GL-Plotparameter zurück.

cam_getintpar - CAM Integerparameter abfragen (CAM)

Synopsis

```
int cam_getintpar(              // Status
      int [0,[;                 // Parametertyp/-nummer:
                                //    0 = Farbcode für Oberste Lage
                                //    1 = Wärmefallenbasiswinkel
                                //    2 = Warnmeldemodus:
                                //        Bit 0: SCM-Änderungs-Warnungen unterdrückt
                                //        Bit 1: im CAM-Prozessor nicht benutzt
                                //        Bit 2: Variantenvergleichswarnungen
                                //                unterdrückt
                                //    3 = Flächenspiegelsicht:
                                //        0 = Standardflächenspiegelsicht
                                //        1 = Flächenspiegelsicht deaktiviert
                                //    4 = Letzter Pixelplotergebnistyp:
                                //        -1 = noch kein Pixelplot
                                //        0 = Pixelverhältnis Platinenumrandung
                                //        1 = Pixelverhältnis Elementgrenzen
                                //    5 = Letzte Pixelplot-Pixelanzahl
                                //    6 = Letzte Pixelplot-Kupferpixelanzahl
                                //    7 = Skalierungsmodus Generische Printausgabe:
                                //        0 = Fester Skalierungsfaktor
                                //        1 = Skalierung auf Papiergröße/span>
                                //    8 = Flag - Farbtabelle gesichert
                                //    9 = Airlinefarbmodus:
                                //        0 = Airlinefarbe benutzen
                                //        1 = Lagenfarbe benutzen
                                //    10 = Bitmapumrandungsfräsmodus:
                                //        0 = Keine Fräsung
                                //        1 = Gefüllte Umrandung fräsen
                                //    11 = Generischer Printer Zeichenmodus:
                                //        0 = Farbe setzen
                                //        1 = Farbe mischen
                                //    12 = Batchausgabeflag
                                //    13 = Flag - Optimierung Bohrwerkzeugtabelle
                                //    14 = Plotvorschaumodus:
                                //        0 = Keine Plotvorschau
                                //        1 = Plotterstiftbreite
      & int;                    // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **cam_getintpar** dient der Abfrage von mit **cam_setintpar** im **CAM-Prozessor** gesetzten Integerparametern. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktions **cam_getdblpar**, **cam_setdblpar**, **cam_setintpar**.

cam_getplotlaycode - CAM Plotlagencode abfragen (CAM)

Synopsis

```
int cam_getplotlaycode(         // HP-GL-Plot-Stiftnummer (CAM4)
      int;                      // Lagennummer (LAY1)
      );
```

Beschreibung

Die Funktion **cam_getplotlaycode** ermittelt die für die spezifizierte Lage aktuell selektierte bzw. gesetzte HP-GL-Plot-Stiftnummer für Multilayer-Plots. Die lagenspezifischen HP-GL-Plot-Stiftnummern werden auch bei der Erzeugung von Multilayer-Plots in anderen Formaten als HP-GL ausgewertet. Positive Stiftnummern kennzeichnen hierbei Lagen, die für die Ausgabe selektiert sind, während Lagen mit negativer Stiftnummer nicht geplottet werden.

Siehe auch

Funktion **cam_setplotlaycode**.

cam_getpowpltparam - CAM Versorgungslagen-Parameter abfragen (CAM)

Synopsis

```
void cam_getpowpltparam(
      & double;              // Min. Distanz Wärmefalle-Bohrung (STD2)
      & double;              // Min. Distanz Isolation-Bohrung (STD2)
      & double;              // Toleranz Wärmefalle-Bohrung (STD2)
      & double;              // Toleranz Isolation-Bohrung (STD2)
      & double;              // Breite Vers.-lagenumrandung (STD2)
      & double;              // Breite Power Plane Isolation (STD2)
      );
```

Beschreibung

Die Funktion **cam_getpowpltparam** gibt die aktuell im **CAM-Prozessor** eingestellten Versorgungslagen-Plotparameter zurück.

cam_getwidedraw - CAM Breitendarstellung abfragen (CAM)

Synopsis

```
double cam_getwidedraw(       // Breite (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **cam_getwidedraw** entspricht der Breite, ab der im **CAM-Prozessor** Leiterbahnen auf dem Bildschirm in Flächendarstellung angezeigt werden.

cam_plotgerber - CAM Gerber-Ausgabe (CAM)

Synopsis

```
int cam_plotgerber(           // Status
      int;                    // Lage (LAY1)
      string;                 // Gerber Dateiname
      double [0.00001,0.01];  // Standardlinienbreite (STD2)
      double ]0.00000000053,[;// Länge einer Ploteinheit (CAM2)
      int [0,1];              // Gerber Optimierung:
                              //    0 = Optimierung aus
                              //    1 = Optimierung ein
      int [0,2];              // Füllverfahren:
                              //    0 = Linien-Füllen
                              //    1 = Multiblenden-Füllen
                              //    2 = G36/G36-Füllmodus
      int [0,1];              // Kreisbogenausgabemodus:
                              //    0 = Kreisbogeninterpolation
                              //    1 = Ausg. mit Gerber I/J-Befehlen
      int [0,2];              // Extended Gerber (RS-274-X) Modus:
                              //    0 = Kein Extended Gerber
                              //    1 = Ext. Gerber mit Standard-Blendentabelle
                              //    2 = Ext. Gerber mit dynamischer Blendentabelle
      int [0,1];              // Error Reset Flag:
                              //    0 = alte Fehler nicht löschen
                              //    1 = alte Fehler löschen
      & int;                  // Anzahl geblitzte Strukturen
      & int;                  // Anzahl rechteckgefüllte Strukturen
      & int;                  // Anzahl kreisgefüllte Strukturen
      & int;                  // Anzahl multigefüllte Strukturen
      & int;                  // Anzahl liniengefüllte Strukturen
      & int;                  // Anzahl gezeichneter Wärmefallen
      & int;                  // Anzahl Überzeichnungsfehler
      );
```

Beschreibung

Die Funktion **cam_plotgerber** erzeugt die Gerber-Plotdaten für die spezifizierte Lage und schreibt die Plotdaten in eine Datei. Wird für den Dateinamen eine Leerzeichenkette übergeben, so werden nur die übergebenen Parameter für Standardlinienbreite, Füllverfahren und Kreisbogen-Ausgabemodus gesetzt und ggf. die Fehleranzeige gelöscht. Der Rückgabewert dieser Funktion ist Null bei fehlerfreiem Plot, 1 bei ungültigen Parametern (Parameter nicht im Wertebereich, keine passende Blende für Standardlinienbreite, ...), und (-1) wenn Plot-Fehler aufgetreten sind. Überzeichnungsfehler werden am Bildschirm hervorgehoben (d.h. mit Highlight) dargestellt.

cam_plothpgl - CAM HP-GL-Ausgabe (CAM)

Synopsis

```
int cam_plothpgl(               // Status
      int;                      // Lage (LAY1)
      int [1,99];               // Stiftnummer
      string;                   // HP-GL Dateiname
      double [0.1,100];         // Maßstab
      double [-1.0,99];         // Geschwindigkeit ([Zentimeter/Sekunde]) oder:
                                //    -1.0 = volle Geschwindigkeit
      double [0.00001,0.01];    // Stiftbreite (STD2)
      int [0,1];                // Füllmodus:
                                //    0 = Füllen aus
                                //    1 = Füllen ein
      int [0,1];                // Error Reset Flag:
                                //    0 = alte Fehler nicht löschen
                                //    1 = alte Fehler löschen
      & int;                    // Rückgabe Anzahl Überzeichnungsfehler
      );
```

Beschreibung

Die Funktion **cam_plothpgl** erzeugt die HP-GL-Plotdaten für die spezifizierte Lage und schreibt die Plotdaten in eine Datei. Wird für den Dateinamen eine Leerzeichenkette übergeben, so werden nur die übergebenen Parameter gesetzt. Der Rückgabewert dieser Funktion ist Null bei fehlerfreiem Plot, und (-1) bei ungültigen Parametern bzw. wenn Plot-Fehler aufgetreten sind. Überzeichnungsfehler werden am Bildschirm hervorgehoben dargestellt.

cam_setdblpar - CAM Doubleparameter setzen (CAM)

Synopsis

```
int cam_setdblpar(              // Status
      int [0,[;                 // Parametertyp/-nummer:
                                //    0 = Wert für dynamische Blendenexpansion
                                //         hinzufügen, 0.0 = Liste löschen (STD2)
                                //    1 = Bitmapplotpixelauflösung (STD2)
                                //    2 = Letztes Bitmapplotpixelverhätnis
      double;                   // Parameterwert
      );
```

Beschreibung

Die Funktion **cam_setdblpar** dient dazu, **CAM-Prozessor**-Systemparameter vom Typ `double` zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **cam_setdblpar** gesetzten Systemparametern können mit der Funktion **cam_getdblpar** abgefragt werden.

Siehe auch

Funktionen **cam_getdblpar**, **cam_getintpar**, **cam_setintpar**.

cam_setdrlaccuracy - CAM Bohrwerkzeugtoleranz setzen (CAM)

Synopsis

```
int cam_setdrlaccuracy(         // Status
      double [0.0,0.01];        // Bohrwerkzeugtoleranz (STD2)
      );
```

Beschreibung

Die Funktion **cam_getdrlaccuracy** setzt den Wert für die Toleranz des Bohrwerkzeugs im **CAM-Prozessor**. Der Funktionsrückgabewert ergibt sich zu Null bei erfolgreicher Einstellung des Toleranzwerts oder zu einem Wert ungleich Null, wenn die Einstellung nicht vorgenommen werden konnte.

Siehe auch

Funktion **cam_getdrlaccuracy**.

cam_setgenpltparam - CAM allgemeine Plotparameter setzen (CAM)

Synopsis

```
int cam_setgenpltparam(       // Status
      int [0,1];              // Alle Lagen Modus:
                              //    0 = Alle Lagen Modus deaktiviert
                              //    1 = Alle Lagen Modus aktiviert
                              //    2 = Plotten alle angeschlossenen Pins/Vias
                              //    3 = Plotten alle Pins und angeschlossene Vias
                              //    4 = Plotten alle Vias und angeschlossene Pins
      int [0,1];              // Umrandungsmodus:
                              //    0 = aus
                              //    1 = ein
      int [0,1];              // Drehung:
                              //    0 = Drehung 0 Grad
                              //    1 = Drehung 90 Grad links
      int [0,5];              // Spiegelung (CAM1)
      int [0,1];              // Passermarkenmodus:
                              //    0 = aus
                              //    1 = ein
      double [0.0,0.01];      // Plotgenauigkeit (STD2)
      double;                 // Plot Nullpunkt X-Koordinate (STD2)
      double;                 // Plot Nullpunkt Y-Koordinate (STD2)
      );
```

Beschreibung

Die Funktion **cam_setgenpltparam** setzt die allgemeinen Plotparameter im **CAM-Prozessor**. Der Rückgabewert der Funktion ist ungleich Null, wenn ungültige Parameter angegeben wurden.

cam_setgerberapt - CAM Gerberblende definieren (CAM)

Synopsis

```
int cam_setgerberapt(         // Status
      int [1,900];            // Blendentabellenindex
      int;                    // Blende D-Code:
                              //    10..999 = gültige D-Codes
                              //       (-1) = Blendendef. löschen
      int [0,3];              // Blendentyp:
                              //    0 = Spezialblende
                              //    1 = Runde Blende
                              //    2 = Quadratische Blende
                              //    3 = Therm. Blende (Wärmefalle)
                              //    4 = Rechteckige Blende
      int [0,2];              // Blendenzeichenmodus:
                              //    0 = Blende für alle Zeichenmodi
                              //    1 = Blende zum Blitzen
                              //    2 = Blende für Linienstrukturen
      double [0.0,[;          // Blendengröße (STD2)
      double [0.0,[;          // Blendengröße 2 (STD2)
      );
```

Beschreibung

Die Funktion **cam_setgerberapt** definiert eine Gerberblende mit den angegebenen Parametern unter dem angegebenen Index der aktuell im **CAM-Prozessor** geladenen Blendentabelle. Wird der Wert (-1) für den D-Code angegeben, dann wird die in der angegebenen Tabellenposition eingetragene Blendendefinition gelöscht. Bei der Definition einer Spezialblende wird die Blendengröße ignoriert. Der Funktionsrückgabewert ist ungleich Null bei fehlenden bzw. ungültigen Parametern.

cam_setintpar - CAM Integerparameter setzen (CAM)

Synopsis

```
int cam_setintpar(              // Status
      int [0,[;                 // Parametertyp/-nummer:
                                //  [ 0 = bae_setcolor benutzen ]
                                //    1 = Wärmefallenbasiswinkel
                                //    2 = Warnmeldemodus:
                                //        Bit 0: SCM-Änderungs-Warnungen
                                //                unterdrücken
                                //        Bit 1: im CAM-Prozessor nicht benutzt
                                //        Bit 2: Variantenvergleichswarnungen
                                //                unterdrücken
                                //  [ 3 = Systemparameter - kein Schreibzugriff ]
                                //  [ 4 = Systemparameter - kein Schreibzugriff ]
                                //  [ 5 = Systemparameter - kein Schreibzugriff ]
                                //  [ 6 = Systemparameter - kein Schreibzugriff ]
                                //    7 = Skalierungsmodus Generische Printausgabe:
                                //        0 = Fester Skalierungsfaktor
                                //        1 = Skalierung auf Papiergröße/span>
                                //    8 = Flag - Farbtabelle gesichert
                                //    9 = Airlinefarbmodus:
                                //        0 = Airlinefarbe benutzen
                                //        1 = Lagenfarbe benutzen
                                //    10 = Bitmapumrandungsfräsmodus:
                                //        0 = Keine Fräsung
                                //        1 = Gefüllte Umrandung fräsen
                                //    11 = Generischer Printer Zeichenmodus:
                                //        0 = Farbe setzen
                                //        1 = Farbe mischen
                                //    12 = Batchausgabeflag
                                //    13 = Flag - Optimierung Bohrwerkzeugtabelle
                                //    14 = Plotvorschaumodus:
                                //        0 = Keine Plotvorschau
                                //        1 = Plotterstiftbreite
      int;                      // Parameterwert
      );
```

Beschreibung

Die Funktion **cam_setintpar** dient dazu, Systemparameter vom Typ im `int` im **CAM-Prozessor** zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **cam_setintpar** gesetzten Systemparametern können mit der Funktion **cam_getintpar** abgefragt werden.

Siehe auch

Funktions **cam_getdblpar**, **cam_getintpar**, **cam_setdblpar**.

cam_setplotlaycode - CAM Plotlagencode setzen (CAM)

Synopsis

```
void cam_setplotlaycode(
      int;                      // Lagennummer (LAY1)
      int;                      // HP-GL-Stiftnummer (CAM4)
      );
```

Beschreibung

Die Funktion **cam_setplotlaycode** selektiert bzw. setzt die angegebene lagenspezifische HP-GL-Plot-Stiftnummer für Multilayer-Plots. Die lagenspezifischen HP-GL-Plot-Stiftnummern werden auch bei der Erzeugung von Multilayer-Plots in anderen Formaten als HP-GL ausgewertet. Positive Stiftnummern kennzeichnen hierbei Lagen, die für die Ausgabe selektiert sind, während Lagen mit negativer Stiftnummer nicht geplottet werden.

Siehe auch

Funktion **cam_getplotlaycode**.

cam_setpowpltparam - CAM Versorgungslagen-Parameter setzen (CAM)

Synopsis

```
int cam_setpowpltparam(       // Status
      double [0.0,0.01];      // Min. Distanz Wärmefalle-Bohrung (STD2)
      double [0.0,0.01];      // Min. Distanz Isolation-Bohrung (STD2)
      double [0.0,0.01];      // Toleranz Wärmefalle-Bohrung (STD2)
      double [0.0,0.01];      // Toleranz Isolation-Bohrung (STD2)
      double [0.0,0.02];      // Breite Vers.-lagenumrandung (STD2)
      double [0.0,0.02];      // Breite Power Plane Isolation (STD2)
      );
```

Beschreibung

Die Funktion **cam_setpowpltparam** setzt die Versorgungslagen-Plotparameter im **CAM-Prozessor**. Der Rückgabewert der Funktion ist ungleich Null, wenn ungültige Parameter angegeben wurden.

cam_setwidedraw - CAM Breitendarstellung setzen (CAM)

Synopsis

```
int cam_setwidedraw(          // Status
      double ]0.0,[;          // Breite (STD2)
      );
```

Beschreibung

Die Funktion **cam_setwidedraw** setzt die Breite, ab der im **CAM-Prozessor** Leiterbahnen auf dem Bildschirm in Flächendarstellung angezeigt werden. Es wird ein Wert ungleich Null zurückgegeben, wenn eine ungültige Breite spezifiziert wurde.

C.4.5 CAM-View-Funktionen

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp CV zugeordnet, d.h. diese Funktionen können im **CAM-View**-Modul aufgerufen werden:

cv_aptgetcolor - CAM-View Blendenfarbe abfragen (CV)

Synopsis

```
int cv_aptgetcolor(            // Farbwert (STD18)
      int;                     // Blendenindex
      int;                     // Blendenmodus
      );
```

Beschreibung

Die Funktion **cv_aptgetcolor** ermittelt den Farbwert, der in **CAM-View** zur Darstellung des angegebenen Gerber-Blendentyps benutzt wird.

Siehe auch

Funktion **cv_aptsetcolor**.

cv_aptsetcolor - CAM-View Blendenfarbe setzen (CV)

Synopsis

```
int cv_aptsetcolor(            // Status
      int;                     // Blendenindex
      int;                     // Blendenmodus
      int [-33554432,33554431];
                               // Farbwert (STD18)
      );
```

Beschreibung

Die Funktion **cv_aptsetcolor** setzt den Farbwert, der in **CAM-View** zur Darstellung des angegebenen Gerber-Blendentyps benutzt werden soll. Der Funktionsrückgabewert ist Null bei erfolgreicher Zuweisung oder ungleich Null andernfalls.

Siehe auch

Funktion **cv_aptgetcolor**.

cv_deldataset - CAM-View Datensatz löschen (CV)

Synopsis

```
int cv_deldataset(             // Status
      int [0,[;                // Datensatz Index
      );
```

Beschreibung

Die Funktion **cv_deldataset** löscht den angebenen **CAM-View**-Datensatz aus dem Arbeitsbereich. Der Funktionsrückgabewert ist Null bei erfolgreicher Ausführung der Operation oder ungleich Null andernfalls.

Siehe auch
Funktion **cv_movedataset**.

cv_getdblpar - CAM-View Doubleparameter abfragen (CV)

Synopsis

```
int cv_getdblpar(              // Status
      int [0,[;                // Parametertyp/-nummer:
                               //    0 = Eingabe X-Offset (STD2)
                               //    1 = Eingabe Y-Offset (STD2)
                               //    2 = Wärmefallenisolationsbreite (STD2)
                               //    3 = Breitendarstellung Startbreite (STD2)
                               //    4 = Länge einer Gerber Plottereinheit (STD2)
      & double;                // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **cv_getdblpar** dient der Abfrage von mit **cv_setdblpar** in **CAM-View** gesetzten Parametern vom Typ **`double`**. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **cv_getintpar**, **cv_setdblpar**, **cv_setintpar**.

cv_getintpar - CAM-View Integerparameter abfragen (CV)

Synopsis

```
int cv_getintpar(              // Status
      int [0,[;                // Parametertyp/-nummer:
                               //    0 = Lagenabfragemodus für Gerber-Import:
                               //        0 = Lage für Line- und Flashstrukturen
                               //             identisch
                               //        1 = Lagenabfrage für Line- und
                               //             Flashstrukturen
                               //    1 = Gerber-Lagenabfrage:
                               //        0 = Lage nicht benutzt
                               //        1 = Lage benutzt
                               //    2 = Farbtabelle/Farbzuweisung:
                               //        0 = Blendenspezifische
                               //             Farbtabelle/Farbzuweisung
                               //        1 = Lagenspezifische
                               //             Farbtabelle/Farbzuweisung
                               //    3 = Flächenbilddarstellungsmodus:
                               //        0 = Füllflächenanzeige
                               //        1 = Umrandungsanzeige
                               //    4 = Via D-Code
                               //    5 = Wärmefallenbasiswinkel
                               //    6 = Gerber Optimierung:
                               //        0 = Koordinatenoptimierung aus
                               //        1 = Koordinatenoptimierung ein
                               //    7 = Gerber Kreisbogenmodus:
                               //        0 = Beliebige Gerber-Kreisbogenwinkel
                               //        1 = Maximal 90 Grad
                               //             Gerber-Kreisbogenwinkel
                               //    8 = Eingabedatenspiegelungsmodus:
                               //        0 = Spiegeln aus
                               //        1 = Spiegeln an X-Achse
                               //        2 = Spiegeln an Y-Achse
                               //        3 = Spiegeln am Ursprung
                               //    9 = Nullzifferunterdrückung:
                               //        0 = Führende Nullen unterdrücken
                               //        1 = Nachfolgende Nullen unterdrücken
                               //    10 = Extended Gerber:
                               //        0 = Extended Gerber aus
                               //        1 = Extended Gerber ein
                               //    11 = Gerber Koordinatenangabe:
                               //        0 = Absolut-Koordinaten
                               //        1 = Inkremental-Koordinaten mit Reset
                               //        2 = Inkremental-Koordinaten ohne Reset
                               //    12 = Gerber Documentarlagenmodus:
                               //        0 = Flashes als Dokumentarlinie
                               //        1 = Flashes als Dokumentarfläche
      & int;                   // Rückgabe Parameterwert
      );
```

Beschreibung

Die Funktion **cv_getintpar** dient der Abfrage von mit **cv_setintpar** in **CAM-View** gesetzten Parametern vom Typ `int`. Der Funktionsrückgabewert ist Null bei erfolgreicher Abfrage oder (-1) im Fehlerfall.

Siehe auch

Funktionen **cv_getdblpar**, **cv_setdblpar**, **cv_setintpar**.

cv_movedataset - CAM-View Datensatz verschieben (CV)

Synopsis

```
int cv_movedataset(             // Status
      int [0,[;                 // Datensatz Index
      int;                      // Datensatz Spiegelungsflag
      double;                   // Datensatz Verschiebung X-Versatz (STD2)
      double;                   // Datensatz Verschiebung Y-Versatz (STD2)
      );
```

Beschreibung

Die Funktion **cv_movedataset** verschiebt (und spiegelt) den angebenen **CAM-View**-Datensatz um den spezifizierten Versatz in X- und Y-Richtung. Der Funktionsrückgabewert ist Null bei erfolgreicher Ausführung der Operation oder ungleich Null andernfalls.

Siehe auch

Funktion **cv_deldataset**.

cv_setdblpar - CAM-View Doubleparameter setzen (CV)

Synopsis

```
int cv_setdblpar(               // Status
      int [0,[;                 // Parametertyp/-nummer:
                                //    0 = Eingabe X-Offset (STD2)
                                //    1 = Eingabe Y-Offset (STD2)
                                //    2 = Wärmefallenisolationsbreite (STD2)
                                //    3 = Breitendarstellung Startbreite (STD2)
                                //    4 = Länge einer Gerber Plottereinheit (STD2)
      double;                   // Parameterwert
      );
```

Beschreibung

Die Funktion **cv_setdblpar** dient dazu, **CAM-View**-Systemparameter vom Typ `double` zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **cv_setdblpar** gesetzten Systemparametern können mit der Funktion **cv_getdblpar** abgefragt werden.

Siehe auch

Funktionen **cv_getdblpar**, **cv_getintpar**, **cv_setintpar**.

cv_setintpar - CAM-View Integerparameter setzen (CV)

Synopsis

```
int cv_setintpar(              // Status
      int [0,[;                // Parametertyp/-nummer:
                               //    0 = Lagenabfragemodus für Gerber-Import:
                               //        0 = Lage für Line- und Flashstrukturen
                               //             identisch
                               //        1 = Lagenabfrage für Line- und
                               //             Flashstrukturen
                               //    1 = Gerber-Lagenbenutzung:
                               //        Parameter nur für Lesezugriff!
                               //    2 = Farbtabelle/Farbzuweisung:
                               //        0 = Blendenspezifische
                               //             Farbtabelle/Farbzuweisung
                               //        1 = Lagenspezifische
                               //             Farbtabelle/Farbzuweisung
                               //    3 = Flächenbilddarstellungsmodus:
                               //        0 = Füllflächenanzeige
                               //        1 = Umrandungsanzeige
                               //    4 = Via D-Code
                               //    6 = Gerber Optimierung:
                               //        0 = Koordinatenoptimierung aus
                               //        1 = Koordinatenoptimierung ein
                               //    7 = Gerber Kreisbogenmodus:
                               //        0 = Beliebige Gerber-Kreisbogenwinkel
                               //        1 = Maximal 90 Grad
                               //             Gerber-Kreisbogenwinkel
                               //    8 = Eingabedatenspiegelungsmodus:
                               //        0 = Spiegeln aus
                               //        1 = Spiegeln an X-Achse
                               //        2 = Spiegeln an Y-Achse
                               //        3 = Spiegeln am Ursprung
                               //    9 = Nullzifferunterdrückung:
                               //        0 = Führende Nullen unterdrücken
                               //        1 = Nachfolgende Nullen unterdrücken
                               //    10 = Extended Gerber:
                               //        0 = Extended Gerber aus
                               //        1 = Extended Gerber ein
                               //    11 = Gerber Koordinatenangabe:
                               //        0 = Absolut-Koordinaten
                               //        1 = Inkremental-Koordinaten mit Reset
                               //        2 = Inkremental-Koordinaten ohne Reset
                               //    12 = Gerber Documentarlagenmodus:
                               //        0 = Flashes als Dokumentarlinie
                               //        1 = Flashes als Dokumentarfläche
      int;                     // Parameterwert
      );
```

Beschreibung

Die Funktion **cv_setintpar** dient dazu, **CAM-View**-Systemparameter vom Typ `int` zu setzen. Der Funktionsrückgabewert ist Null bei erfolgreicher Parameterzuweisung oder (-1) im Fehlerfall. Die Werte von mit **cv_setintpar** gesetzten Systemparametern können mit der Funktion **cv_getintpar** abgefragt werden.

Siehe auch

Funktionen **cv_getdblpar**, **cv_getintpar**, **cv_setdblpar**.

C.5 IC-Design-Systemfunktionen

In diesem Abschnitt werden (in alphabetischer Reihenfolge) die in der **Bartels User Language** definierten **IC-Design**-Systemfunktionen beschrieben. Beachten Sie bitte die Konventionen zur Funktionsbeschreibung in Anhang C.1.

C.5.1 IC-Design-Datenzugriffsfunktionen

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp ICD zugeordnet, d.h. diese Funktionen können im **Chipeditor** aufgerufen werden:

icd_altpinlay - IC Design Setup Alternativpinlayer (ICD)

Synopsis

```
int icd_altpinlay(             // Alternativpinlayer (ICD1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_altpinlay** entspricht der in der Setupdatei angegebenen Einstellung für den in der GDS-Dateneingabe verwendeten Alternativpinlayer.

icd_cellconlay - IC Design Setup Lage interne Zellverbindungen (ICD)

Synopsis

```
int icd_cellconlay(            // Zellverbindungslayer (ICD1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_cellconlay** entspricht der in der Setupdatei angegebenen Einstellung für den beim automatischen Routen zur Herstellung interner Zellverbindungen verwendeten Layer.

icd_cellscan - IC Design Setup DRC auf Zellebene (ICD)

Synopsis

```
int icd_cellscan(              // Zell DRC-Modus
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_cellscan** entspricht der in der Setupdatei angegebenen Einstellung für den DRC auf Zellebene (0 = kein DRC für Zellstrukturen, 1 = DRC für Zellstrukturen).

icd_cellshr - IC Design Setup Zellsperrflächenoffset (ICD)

Synopsis

```
double icd_cellshr(            // Zellsperrflächenoffset (STD2)
      );
```

Beschreibung
Der Rückgabewert der Funktion **icd_cellshr** entspricht der in der Setupdatei angegebenen Einstellung für die Flächenverkleinerung für die automatische Generierung von Zellsperrflächen aus der Zellumrandung.

icd_ciflayname - IC Design CIF-Ausgabelage abfragen (ICD)

Synopsis

```
string icd_ciflayname(          // Lagenname
      int [0,99];               // Lagennummer (ICD1)
      );
```

Beschreibung

Die Funktion **icd_ciflayname** ermittelt die in der Setupdatei angegebene CIF-Ausgabelagenbezeichnung für die angegebene Lage (ICD1).

icd_cstdsiz - IC Design Setup Standardzellenhöhe abfragen (ICD)

Synopsis

```
double icd_cstdsiz(             // Standardzellenhöhe (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_cstdsiz** entspricht der in der Setupdatei angegebenen Einstellung für die vom automatischen Zellplatzierer zu verwendende Höhe von Standardzellen.

icd_defelemname - IC Design Setup default Elementname (ICD)

Synopsis

```
string icd_defelemname(         // Default Layout Elementname
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_defelemname** entspricht dem in der Setupdatei eingestellten Defaultnamen für Chip Layoutplanelemente.

icd_deflibname - IC Design Setup default Bibliothek (ICD)

Synopsis

```
string icd_deflibname(          // Default IC Design Bibliotheksname
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_deflibname** entspricht dem in der Setupdatei eingestellten Defaultnamen für die **IC-Design**-Bibliothek.

icd_drcarc - IC Design Setup DRC Kreisbögen abfragen (ICD)

Synopsis

```
int icd_drcarc(                 // DRC Kreisbogenmodus
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_drcarc** entspricht der in der Setupdatei angegebenen Einstellung für den Design Rule Check (DRC) für Kreisbögen (0 = Kreisbögen erlaubt, 1 = keine Kreisbögen erlaubt).

icd_drcgrid - IC Design Setup DRC Raster abfragen (ICD)

Synopsis

```
double icd_drcgrid(             // DRC Raster (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_drcgrid** entspricht der in der Setupdatei angegebenen Einstellung für das durch den Design Rule Check (DRC) einzuhaltende Raster.

icd_drclaymode - IC Design Setup DRC Lagenberücksichtigung (ICD)

Synopsis

```
int icd_drclaymode(             // Lage DRC-Modus
      int [0,99];               // Lagennummer (ICD1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_drclaymode** entspricht der in der Setupdatei angegebenen Einstellung für den Design Rule Check (DRC) auf der angegebenen Lage (0 = kein DRC auf angegebener Lage, 1 = DRC auf angegebener Lage).

icd_drcmaxpar - IC Design Setup DRC Parallelcheck abfragen (ICD)

Synopsis

```
double icd_drcmaxpar(           // DRC maximale Parallellänge (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_drcmaxpar** entspricht der in der Setupdatei angegebenen Einstellung für die beim Design Rule Check (DRC) maximal zulässige Länge paralleler Strukturen.

icd_drcminwidth - IC Design Setup DRC minimale Strukturgröße (ICD)

Synopsis

```
double icd_drcminwidth(         // DRC minimale Strukturgröße (STD2)
      int [0,99];               // Lagennummer (ICD1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_drcminwidth** entspricht der in der Setupdatei angegebenen Einstellung für die beim Design Rule Check (DRC) minimal zulässige Strukturgröße für die angegebene Lage. Ein negativer Rückgabewert bedeutet, dass nur quadratische Strukturen mit exakt der angegebenen Größe erlaubt sind.

icd_drcrect - IC Design Setup DRC Orthogonalcheck abfragen (ICD)

Synopsis

```
int icd_drcrect(                // DRC rechte Winkel Modus
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_drcrect** entspricht der in der Setupdatei angegebenen Einstellung für den Design Rule Check (DRC) für rechte Winkel (0 = beliebige Winkel erlaubt, 1 = nur rechte Winkel erlaubt).

icd_ecnlaymode - IC Design Setup Lagenconnectivity (ICD)

Synopsis

```
int icd_ecnlaymode(             // Lage Connectivity-Modus
      int [0,99];               // Lagennummer (ICD1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_ecnlaymode** entspricht der in der Setupdatei angegebenen Einstellung für die Connectivity auf der angegebenen Lage (0 = keine Connectivity auf der angegebenen Lage, 1 = Connectivity auf der angegebener Lage).

icd_findconpart - IC Design Bauteil in Netzliste suchen (ICD)

Synopsis

```
int icd_findconpart(           // Status
      string;                  // Bauteilname
      & index I_CPART;         // Rückgabe Netzlistenbauteil
      );
```

Beschreibung

Die Funktion **icd_findconpart** sucht den angegebenen Bauteilnamen in der Netzliste und gibt den Bauteileintrag gegebenenfalls in dem Bauteilrückgabeparameter zurück. Der Rückgabewert dieser Funktion ist ungleich Null, wenn das Bauteil nicht gefunden wurde.

Siehe auch

Funktionen **icd_findconpartpin**, **icd_findcontree**.

icd_findconpartpin - IC Design Bauteilpin in Netzliste suchen (ICD)

Synopsis

```
int icd_findconpartpin(        // Status
      string;                  // Pinname
      index I_CPART;           // Netzlistenbauteil
      & index I_CPIN;          // Rückgabe Netzlistenbauteilpin
      );
```

Beschreibung

Die Funktion **icd_findconpartpin** sucht den Bauteilpin mit dem angegebenen Namen auf dem spezifizierten Netzlistenbauteil und gibt den Bauteilpineintrag gegebenenfalls in dem Bauteilpinrückgabeparameter zurück. Der Rückgabewert dieser Funktion ist ungleich Null, wenn der Bauteilpin nicht gefunden wurde.

Siehe auch

Funktionen **icd_findconpart**, **icd_findcontree**.

icd_findcontree - IC Design Netz in Netzliste suchen (ICD)

Synopsis

```
int icd_findcontree(           // Status
      string;                  // Netzname
      & index I_CNET;          // Rückgabe Netzlisteneintrag
      );
```

Beschreibung

Die Funktion **icd_findcontree** sucht den angegebenen Netznamen in der Netzliste und gibt den Netzlisteneintrag ggf. in dem Netzrückgabeparameter zurück. Der Rückgabewert dieser Funktion ist ungleich Null, wenn das Netz nicht gefunden wurde.

Siehe auch
Funktionen **icd_findconpart**, **icd_findconpartpin**.

icd_getrulecnt - IC Design-Element Regelanzahl abfragen (ICD)

Synopsis

```
int icd_getrulecnt(           // Regelanzahl oder (-1) bei Fehler
      int;                    // Object class code
      int;                    // Object ident code (int oder Indextyp)
      );
```

Beschreibung

Mit der Funktion **icd_getrulecnt** kann die Anzahl der an ein spezifisches Objekt zugewiesenen Regeln ermittelt werden. Die Abfrage kann für das aktuell geladene Element (Objektklasse 0 mit `int`-Wert 0 für die Objektidentifikation), ein Element aus der Figurenliste des aktuell geladenen Elements (Objektklasse 1 mit gültigem Wert vom Typ index **I_FIGURE** für die Objektidentifikation) oder ein Poolelement (Objektklasse 2 mit gültigem Wert vom Typ index **I_POOL** für die Objektidentifikation) durchgeführt werden. Die von **icd_getrulecnt** ermittelte (nicht-negative) objektspezifische Regelanzahl wird im Rückgabewert der Funktion übergeben und bestimmt den Wertebereich für den Regelnamenslistenindex in nachfolgenden Aufrufen der Funktion **icd_getrulename** zur Ermittlung von Regelnamen für das entsprechende Objekt. Der Rückgabewert ergibt sich zu (-1), wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **icd_ruleerr** ermittelt werden.

Siehe auch

Funktionen **icd_getrulename**, **icd_ruleerr**, **icd_rulefigatt**, **icd_rulefigdet**, **icd_ruleplanatt**, **icd_ruleplandet**, **icd_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

icd_getrulename - IC Design-Element Regelname abfragen (ICD)

Synopsis

```
int icd_getrulename(          // Status
      int;                    // Object class code
      int;                    // Object ident code (int oder Indextyp)
      int [0,[;               // Regelnamenslistenindex
      & string;               // Regelname
      );
```

Beschreibung

Mit der Funktion **icd_getrulename** können die Namen der an ein spezifisches Objekt zugewiesenen Regeln ermittelt werden. Die Abfrage kann für das aktuell geladene Element (Objektklasse 0 mit `int`-Wert 0 für die Objektidentifikation), ein Element aus der Figurenliste des aktuell geladenen Elements (Objektklasse 1 mit gültigem Wert vom Typ index **I_FIGURE** für die Objektidentifikation) oder ein Poolelement (Objektklasse 2 mit gültigem Wert vom Typ index **I_POOL** für die Objektidentifikation) durchgeführt werden. Der Regelnamenslistenindex zur Auswahl der gewünschten Regel muss mindestens Null jedoch kleiner als die mit der Funktion **icd_getrulecnt** abfragbare Anzahl objektspezifischer Regeln sein. Der ermittelte Regelname wird über den letzten Funktionsparameter an den Aufrufer zurückgegeben. Der Rückgabewert der Funktion **icd_getrulename** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **icd_ruleerr** ermittelt werden.

Siehe auch

Funktionen **icd_getrulecnt**, **icd_ruleerr**, **icd_rulefigatt**, **icd_rulefigdet**, **icd_ruleplanatt**, **icd_ruleplandet**, **icd_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

icd_gettreeidx - IC Design Netznummer in Netzliste suchen (ICD)

Synopsis

```
int icd_gettreeidx(           // Status
      int;                    // Netznummer
      & index I_CNET;         // Rückgabe Netzlisteneintrag
      );
```

Beschreibung

Die Funktion **icd_gettreeidx** sucht die angegebene Netznummer in der Netzliste und gibt den Netzlisteneintrag gegebenenfalls in dem Netzrückgabeparameter zurück. Der Rückgabewert ist ungleich Null, wenn die Netznummer nicht gefunden wurde.

icd_grpdisplay - IC Design Setup Gruppenlage abfragen (ICD)

Synopsis

```
int icd_grpdisplay(              // Lagennummer (ICD1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_grpdisplay** entspricht der in der Setupdatei eingestellten Lage für die Gruppendarstellung.

icd_lastfigelem - Zuletzt modifiziertes IC Design Element ermitteln (ICD)

Synopsis

```
int icd_lastfigelem(             // Status
      & index I_FIGURE;          // Rückgabe Element
      );
```

Beschreibung

Die Funktion **icd_lastfigelem** ermittelt das zuletzt erzeugte bzw. modifizierte IC Design Element und übergibt den entsprechenden Index aus der Figurenliste im Rückgabeparameter. Der Rückgabewert der Funktion ist Null wenn ein derartiges Element existiert, oder ungleich Null andernfalls.

icd_maccoords - IC Design Makrokoordinaten abfragen (ICD)

Synopsis

```
void icd_maccoords(
      & double;                  // X-Position (STD2)
      & double;                  // Y-Position (STD2)
      & double;                  // Drehwinkel (STD3)
      & double;                  // Skalierungsfaktor
      & int;                     // Spiegelungsmodus (STD14)
      );
```

Beschreibung

Die Funktion **icd_maccoords** gibt in den Parametern die Platzierungsdaten für das aktuell bearbeitete Makro zurück. Ein Aufruf dieser Funktion ist nur innerhalb der Makroscanfunktion von **icd_scanall**, **icd_scanfelem**, **icd_scanpool** sinnvoll. An anderer Stelle werden Null-Defaultwerte zurückgegeben.

Siehe auch

Funktionen **icd_scanall**, **icd_scanfelem**, **icd_scanpool**.

icd_nrefsearch - IC Design Name auf Plan suchen (ICD)

Synopsis

```
int icd_nrefsearch(              // Status
      string;                    // Bauteilname
      & index I_FIGURE;          // Rückgabe Element
      );
```

Beschreibung

Die Funktion **icd_nrefsearch** prüft, ob das angegebene Bauteil platziert ist und gibt gegebenenfalls das zugehörige Element zurück. Der Rückgabewert ist ungleich Null, wenn das Bauteil nicht gefunden wurde.

icd_outlinelay - IC Design Setup Zellumrandung Lage abfragen (ICD)

Synopsis

```
int icd_outlinelay(            // Zellumrandungslage (ICD1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_outlinelay** entspricht der in der Setupdatei angegebenen Einstellung für die Zellumrandungslage.

icd_pindist - IC Design Setup Pinaussparung abfragen (ICD)

Synopsis

```
double icd_pindist(            // Pinaussparung (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_pindist** entspricht der in der Setupdatei angegebenen Einstellung für die Aussparung von Pins bei der automatischen Generierung von Zellsperrflächen aus der Zellumrandung.

icd_plcxgrid - IC Design Setup Platzierungsraster abfragen (ICD)

Synopsis

```
double icd_plcxgrid(           // Platzierungsraster (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_plcxgrid** entspricht der in der Setupdatei angegebenen Einstellung für das Platzierungsraster in X-Richtung bei der automatischen Zellplatzierung.

icd_plcxoffset - IC Design Setup Platzierungsoffset abfragen (ICD)

Synopsis

```
double icd_plcxoffset(         // Platzierungsoffset (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_plcxoffset** entspricht der in der Setupdatei angegebenen Einstellung für den Platzierungsoffset in X-Richtung bei der automatischen Zellplatzierung.

icd_routcellcnt - IC Design Setup Anzahl Stromversorgungszellen (ICD)

Synopsis

```
int icd_routcellcnt(           // Anzahl Stromversorgungszellen
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_routcellcnt** entspricht der Anzahl der in der Setupdatei angegebenen Stromversorgungszellen.

icd_routcellname - IC Design Standardlayer Name abfragen (ICD)

Synopsis

```
string icd_routcellname(       // Lagenname
      int [0,[;                 // Zellindex
      );
```

Beschreibung

Die Funktion **icd_routcellname** ermittelt die in der Setupdatei unter dem angegebenen Index aufgeführte Stromversorgungszelle. Der übergebene Zellindex darf im Bereich 0 bis **icd_routcellcnt**()-1 liegen.

icd_ruleerr - Regelsystem Fehlerstatus abfragen (ICD)

Synopsis

```
void icd_ruleerr(
      & int;             // Fehlercode
      & string;          // Fehlerstring
      );
```

Beschreibung

Die Funktion **icd_ruleerr** dient der Ermittlung des Regelsystemstatus, d.h. die Funktion **icd_ruleerr** kann zur genauen Bestimmung der Fehlerursache im Falle eines fehlerhaften Aufrufs einer Regelsystemfunktion verwendet werden.

Diagnose

Zur Bestimmung der Fehlerursache sind die durch **icd_ruleerr** zurückgegebenen Parameterwerte heranzuziehen. Der zurückgegebene Fehlerstring dient ggf. der Identifizierung des fehlerverursachenden Elements. Die möglichen Werte, die der Fehlercode durch die Ausführung eines Regelsystemfunktion annehmen kann, haben folgende Bedeutung:

Fehlercode	Bedeutung
0	Regelsystem Operation/Funktion erfolgreich beendet
1	Regelsystem Hauptspeicher nicht ausreichend
2	Regelsystem Interner Fehler **<e>**
3	Regelsystem Funktionsparameter ungültig
128	Regelsystem Datenbankdatei kann nicht angelegt werden
129	Regelsystem Datenbankdatei Lese-/Schreibfehler
130	Regelsystem Datenbankdatei von falschem Typ
131	Regelsystem Datenbankdateistruktur beschädigt
132	Regelsystem Datenbankdatei nicht gefunden
133	Regelsystem Datenbankfehler allgemein (Interner Fehler)
134	Regelsystem Regel **<r>** nicht Regeldatenbank gefunden
135	Regelsystem Regel in falschem Format in Datenbank (Interner Fehler **<e>**)
136	Regelsystem Objekt nicht gefunden
137	Regelsystem Objekt mehrfach definiert (Interner Fehler)
138	Regelsystem Inkompatible Definition der Variable **<v>**
139	Regelsystem Regel **<r>** mit inkompatibler Compiler-Version übersetzt

Der Fehlerstring kann je nach Fehlerfall eine Regel **<r>**, eine Variable **<v>** oder einen (internen) Fehlerstatus **<e>** bezeichnen. Datenbankdateifehler beziehen sich auf Probleme beim Zugriff auf die Regeldatenbankdatei **`brules.vdb`** im BAE-Programmverzeichnis. Interne Fehler weisen üblicherweise auf Implementierungslücken im Regelsystem hin und sollten in jedem Fall an Bartels gemeldet werden.

Siehe auch

Funktionen **icd_getrulecnt**, **icd_getrulename**, **icd_rulefigatt**, **icd_rulefigdet**, **icd_ruleplanatt**, **icd_ruleplandet**, **icd_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

icd_rulefigatt - Regelzuweisung an Figurenelement (ICD)

Synopsis

```
int icd_rulefigatt(              // Status
      index I_FIGURE;            // Figurenlistenelement
      void;                      // Regelname oder Regelnamensliste
      );
```

Beschreibung

Die Funktion **icd_rulefigatt** erlaubt die Zuweisung von Regeln an das mit dem ersten Funktionsparameter spezifizierte Figurenlistenelement des aktuell geladenes Elements. Der zweite Funktionsparameter erlaubt dabei sowohl die Spezifikation eines einzelnen Regelnamens (d.h. eines Wertes vom Typ `string`) als auch die Angabe einer ganzen Liste von Regelnamen (d.h. eines Arrays vom Typ `string`). Beachten Sie, dass vor der Zuweisung des angegebenen Regelsatzes zunächst alle bestehenden Regelzuweisungen an das spezifizierte Figurenlistenelement gelöscht werden. Der Rückgabewert der Funktion **icd_rulefigatt** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **icd_ruleerr** ermittelt werden.

Siehe auch

Funktionen **icd_getrulecnt**, **icd_getrulename**, **icd_ruleerr**, **icd_rulefigdet**, **icd_ruleplanatt**, **icd_ruleplandet**, **icd_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

icd_rulefigdet - Regelzuweisungen von Figurenelement lösen (ICD)

Synopsis

```
int icd_rulefigdet(              // Status
      index I_FIGURE;            // Figurenlistenelement
      );
```

Beschreibung

Die Funktion **icd_rulefigdet** löscht *alle* aktuell bestehenden Regelzuweisungen an das über den Funktionsparameter spezifizierte Figurenlistenelement des aktuell geladenen Elements. Der Rückgabewert der Funktion **icd_rulefigdet** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **icd_ruleerr** ermittelt werden.

Siehe auch

Funktionen **icd_getrulecnt**, **icd_getrulename**, **icd_ruleerr**, **icd_rulefigatt**, **icd_ruleplanatt**. **icd_ruleplandet**, **icd_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

icd_ruleplanatt - Regelzuweisung an aktuell geladenes Element (ICD)

Synopsis

```
int icd_ruleplanatt(             // Status
      void;                      // Regelname oder Regelnamensliste
      );
```

Beschreibung

Die Funktion **icd_ruleplanatt** erlaubt die Zuweisung von Regeln an das aktuell geladenes Elements. Der Funktionsparameter erlaubt dabei sowohl die Spezifikation eines einzelnen Regelnamens (d.h. eines Wertes vom Typ `string`) als auch die Angabe einer ganzen Liste von Regelnamen (d.h. eines Arrays vom Typ `string`). Beachten Sie, dass vor der Zuweisung des angegebenen Regelsatzes zunächst alle bestehenden Regelzuweisungen an das aktuelle Element gelöscht werden. Der Rückgabewert der Funktion **icd_ruleplanatt** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **icd_ruleerr** ermittelt werden.

Siehe auch

Funktionen **icd_getrulecnt**, **icd_getrulename**, **icd_ruleerr**, **icd_rulefigatt**, **icd_rulefigdet**, **icd_ruleplandet**, **icd_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

icd_ruleplandet - Regelzuweisungen von aktuell geladenem Element lösen (ICD)

Synopsis

```
int icd_ruleplandet(          // Status
      );
```

Beschreibung

Die Funktion **icd_ruleplandet** löscht *alle* aktuell bestehenden Regelzuweisungen an das aktuell geladene Element. Der Rückgabewert der Funktion **icd_ruleplandet** ist Null, wenn die Funktion erfolgreich beendet wurde oder ungleich Null, wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **icd_ruleerr** ermittelt werden.

Siehe auch

Funktionen **icd_getrulecnt**, **icd_getrulename**, **icd_ruleerr**, **icd_rulefigatt**, **icd_rulefigdet**, **icd_ruleplanatt**, **icd_rulequery**; **Neuronales Regelsystem** und Rule System Compiler.

icd_rulequery - IC Design-Element Regelabfrage durchführen (ICD)

Synopsis

```
int icd_rulequery(             // Trefferanzahl oder (-1) bei Fehler
      int;                     // Object class code
      int;                     // Object ident code (int oder Indextyp)
      string;                  // Subjektname
      string;                  // Prädikatname
      string;                  // Abfragekommando
      & void;                  // Abfrageergebnis
      []                       // Optionale Abfrageparameter
      );
```

Beschreibung

Die Funktion **icd_rulequery** führt eine Regelabfrage für ein spezifisches Objekt durch. Die Abfrage kann für das aktuell geladene Element (Objektklasse 0 mit **int**-Wert 0 für die Objektidentifikation), ein Element aus der Figurenliste des aktuell geladenen Elements (Objektklasse 1 mit gültigem Wert vom Typ index **I_FIGURE** für die Objektidentifikation) oder ein Poolelement (Objektklasse 2 mit gültigem Wert vom Typ index **I_POOL** für die Objektidentifikation) durchgeführt werden. Zur Durchführung der Abfrage müssen sowohl ein Regelsubjekt als auch ein Regelprädikat namentlich angegeben werden. Zusätzlich ist ein Abfragekommando zu spezifizieren. Das Abfragekommando kann Platzhalter für Wertvorgaben und einen Abfrageoperator enthalten. Folgende Abfrageoperatoren stehen zur Verfügung:

?d	zur Abfrage von **int**-Werten
?f	zur Abfrage von **double**-Werten
?s	zur Abfrage von **string**-Werten

Dem Abfrageoperator kann wahlweise einer der folgenden Selektionsoperatoren vorangestellt werden:

+	zur Abfrage des Maximums aller gefundenen Werte
-	zur Abfrage des Minimums aller gefundenen Werte

Standardmäßig, d.h. bei Auslassung des Selektionsoperators wird der **+**-Operator verwendet. Der über die Abfrage gefundene Werteintrag wird im Funktionsparameter für das Abfrageergebnis zurückgegeben. Hierbei ist sicherzustellen, dass der Datentyp des Parameters für das Abfrageergebnis mit dem Abfragedatentyp übereinstimmt (**int** für **?d**, **double** für **?f**, **string** für **?s**). Neben dem Abfrageoperator können folgende Platzhalter für Wertvorgaben im Abfragekommando spezifiziert werden:

%d	zur Angabe von **int**-Werten
%f	zur Angabe von **double**-Werten
%s	zur Angabe von **string**-Werten

Für jeden im Abfragekommando spezifizierten Platzhalter für Wertvorgaben ist ein optionaler Abfrageparameter an die Funktion **icd_rulequery** zu übergeben. Die Reihenfolge dieser optionalen Parameter sowie deren Datentypen müssen mit den Spezifikationen im Abfragekommando übereinstimmen. Nach erfolgreicher Abarbeitung der Regelabfrage wird im Rückgabewert die (nicht-negative) Anzahl der gefundenen Einträge an den Aufrufer zurückgegeben. Der Rückgabewert ergibt sich zu (-1), wenn ein Fehler aufgetreten ist. Im Fehlerfall kann die genaue Fehlerursache mit Hilfe der Funktion **icd_ruleerr** ermittelt werden.

Beispiele

Sofern die Regel

```
rule somerule
{
    subject subj
    {
        pred := ("A", 2);
        pred := ("A", 4);
        pred := ("B", 1);
        pred := ("C", 3);
        pred := ("B", 6);
        pred := ("D", 5);
        pred := ("D", 6);
        pred := ("A", 3);
    }
}
```

definiert und dem aktuell geladenen Element zugewiesen ist, würde der **icd_rulequery**-Aufruf

```
hitcount = icd_rulequery(0,0,"subj","pred","%s ?d",intresult,"A") ;
```

die **`int`**-Variable **`hitcount`** auf 3 und die **`int`**-Variable **`intresult`** auf 4 setzen, während der Aufruf

```
hitcount = icd_rulequery(0,0,"subj","pred","-?s %d",strresult,6) ;
```

die Variable **`hitcount`** auf 2 und die **`string`**-Variable **`strresult`** auf **`B`** setzt.

Siehe auch

Funktionen **icd_getrulecnt**, **icd_getrulename**, **icd_ruleerr**, **icd_rulefigatt**, **icd_rulefigdet**, **icd_ruleplanatt**, **icd_ruleplandet**; **Neuronales Regelsystem** und Rule System Compiler.

icd_scanall - IC Design Scan über alle Elemente (ICD)

Synopsis

```
int icd_scanall(                // Scan Status
      double;                   // X-Offset (STD2)
      double;                   // Y-Offset (STD2)
      double;                   // Drehwinkel (STD3)
      int [0,1];                // Element in Arbeitsbereich Flag (STD10)
      int [0,1];                // Connectivity Scan Flag:
                                //    0 = kein Scan
                                //    1 = Scan erlaubt
      * int;                    // Makrofunktion
      * int;                    // Polygonfunktion
      * int;                    // Leiterbahnfunktion
      * int;                    // Textfunktion
      * int;                    // Lagencheckfunktion
      * int;                    // Levelcheckfunktion
      );
```

Beschreibung

Die Funktion **icd_scanall** scannt alle auf dem aktuell geladenen Element platzierten Elemente über alle Hierarchiestufen. Dabei werden für alle gefundenen Elemente die referenzierten Anwenderfunktionen aufgerufen. Soll ein Funktionstyp nicht aufgerufen werden, so ist für den entsprechenden Parameter **NULL** anzugeben. Der Rückgabewert der Funktion **icd_scanall** ist ungleich Null, wenn ungültige Parameter angegeben wurden, oder wenn eine der referenzierten Anwenderfunktionen einen Fehler an die Funktion **icd_scanall** zurückgemeldet hat.

Makrofunktion

```
int macrofuncname(
      index I_MACRO macro,      // Makro Index
      index I_POOL pool,        // Pool Element Index
      int macinws,              // Makro in Arbeitsbereich Flag (STD10)
      string refname,           // Makro Referenzname
      index I_LEVEL level,      // Makro Level
      int stkcnt                // Makro Stacktiefe
      )
{
      // Verarbeitungsprogramm
      :
      return(contscan);
}
```

Die Makroplatzierungsdaten können mit der Funktion **icd_maccoords** abgefragt werden. Der Rückgabewert dieser Funktion sollte gleich Null sein, wenn der Scan für dieses Makro nicht weitergeführt werden soll, (-1) wenn ein Fehler aufgetreten ist und 1 wenn weitergescannt werden soll.

Polygonfunktion

```
int polyfuncname(
      index I_POLY poly,        // Polygondaten
      int layer,                // Lage (ICD1)
      int polyinws,             // Polygon in Arbeitsbereich Flag (STD10)
      int tree,                 // Netznummer oder (-1)
      index I_LEVEL level       // Polygon Level
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Der Rückgabewert dieser Funktion sollte ungleich Null sein, wenn ein Fehler aufgetreten ist.

Leiterbahnfunktion

```
int pathfuncname(
      index I_LINE path,       // Leiterbahndaten
      int layer,               // Lage (ICD1)
      int pathinws,            // Bahn in Arbeitsbereich Flag (STD10)
      index I_LEVEL level      // Bahn Level
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Der Rückgabewert dieser Funktion sollte ungleich Null sein, wenn ein Fehler aufgetreten ist.

Textfunktion

```
int textfuncname(
      index I_TEXT text,       // Textdaten
      double x,                // X-Koordinate (STD2)
      double y,                // Y-Koordinate (STD2)
      double angle,            // Drehwinkel (STD3)
      int mirr,                // Spiegelung (STD14)
      int layer,               // Lage (ICD1)
      double size,             // Text Größe (STD2)
      string textstr,          // Textzeichenkette
      int textinws             // Text in Arbeitsbereich Flag (STD10)
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Der Rückgabewert dieser Funktion sollte ungleich Null sein, wenn ein Fehler aufgetreten ist.

Lagencheckfunktion

```
int laycheckfuncname(
      int layer,               // Lage (ICD1)
      int class                // Elementklasse (STD1)
      )
{
      // Verarbeitungsprogramm
      :
      return(contscan);
}
```

Der Rückgabewert dieser Funktion sollte gleich Null sein, wenn der Scan bei der übergebenen Lage nicht weitergeführt werden soll, (-1) wenn ein Fehler aufgetreten ist und 1 wenn weitergescannt werden soll. Bei Beschränkung auf die interessierenden Lagen durch diese Funktion wird der Scanvorgang erheblich beschleunigt.

Levelcheckfunktion

```
int levcheckfuncname(
      index I_LEVEL level     // Level
      )
{
      // Verarbeitungsprogramm
      :
      return(contscan);
}
```

Der Rückgabewert dieser Funktion sollte gleich Null sein, wenn der Scan bei dem übergebenen Level nicht weitergeführt werden soll, (-1) wenn ein Fehler aufgetreten ist und 1 wenn weitergescannt werden soll. Bei Beschränkung auf die interessierenden Levels durch diese Funktion wird der Scanvorgang erheblich beschleunigt.

Siehe auch

Funktionen **icd_maccoords**, **icd_scanfelem**, **icd_scanpool**.

icd_scanfelem - IC Design Scan über Figurenelement (ICD)

Synopsis

```
int icd_scanfelem(            // Scan Status
      index I_FIGURE;         // Figurenelement
      double;                 // X-Offset (STD2)
      double;                 // Y-Offset (STD2)
      double;                 // Drehwinkel (STD3)
      int [0,1];              // Element in Arbeitsbereich Flag (STD10)
      int [0,1];              // Connectivity Scan Flag:
                              //    0 = kein Scan
                              //    1 = Scan erlaubt
      * int;                  // Makrofunktion
      * int;                  // Polygonfunktion
      * int;                  // Leiterbahnfunktion
      * int;                  // Textfunktion
      * int;                  // Lagencheckfunktion
      * int;                  // Levelcheckfunktion
      );
```

Beschreibung

Die Funktion **icd_scanfelem** scannt das angegebene Figurenelement über alle Hierarchiestufen. Dabei werden für alle gefundenen Elemente die referenzierten Anwenderfunktionen aufgerufen. Soll ein Funktionstyp nicht aufgerufen werden, so ist für den entsprechenden Parameter **NULL** anzugeben (Definition der referenzierten Anwenderfunktionen siehe **icd_scanall**). Der Rückgabewert der Funktion **icd_scanfelem** ist ungleich Null, wenn ungültige Parameter angegeben wurden, oder wenn eine der referenzierten Anwenderfunktionen einen Fehler an die Funktion **icd_scanfelem** zurückgemeldet hat.

Siehe auch

Funktionen **icd_maccoords**, **icd_scanall**, **icd_scanpool**.

icd_scanpool - IC Design Scan über Poolelement (ICD)

Synopsis

```
int icd_scanpool(               // Scan Status
      void;                     // Poolelement
      double;                   // X-Offset (STD2)
      double;                   // Y-Offset (STD2)
      double;                   // Drehwinkel (STD3)
      int [0,1];                // Element in Arbeitsbereich Flag (STD10)
      int [0,1];                // Connectivity Scan Flag:
                                //    0 = kein Scan
                                //    1 = Scan erlaubt
      * int;                    // Makrofunktion
      * int;                    // Polygonfunktion
      * int;                    // Leiterbahnfunktion
      * int;                    // Textfunktion
      * int;                    // Lagencheckfunktion
      * int;                    // Levelcheckfunktion
      );
```

Beschreibung

Die Funktion **icd_scanpool** scannt das angegebene Poolelement über alle Hierarchiestufen. Dabei werden für alle gefundenen Elemente die referenzierten Anwenderfunktionen aufgerufen. Soll ein Funktionstyp nicht aufgerufen werden, so ist für den entsprechenden Parameter **NULL** anzugeben (Definition der referenzierten Anwenderfunktionen siehe **icd_scanall**). Der Rückgabewert der Funktion **icd_scanpool** ist ungleich Null, wenn ungültige Parameter angegeben wurden, oder wenn eine der referenzierten Anwenderfunktionen einen Fehler an die Funktion **icd_scanpool** zurückgemeldet hat.

Siehe auch

Funktionen **icd_maccoords**, **icd_scanall**, **icd_scanfelem**.

icd_stdlayname - IC Design Standardlayer Name abfragen (ICD)

Synopsis

```
string icd_stdlayname(          // Lagenname
      int [0,99];               // Lagennummer (ICD1)
      );
```

Beschreibung

Die Funktion **icd_stdlayname** ermittelt die in der Setupdatei angegebene Lagenbezeichnung für die angegebene Lage (ICD1).

icd_stdpinlay - IC Design Setup Standardpinlayer (ICD)

Synopsis

```
int icd_stdpinlay(              // Standardpinlayer (ICD1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **icd_stdpinlay** entspricht der in der Setupdatei angegebenen Einstellung für den in der GDS-Dateneingabe verwendeten Standardpinlayer.

icd_vecttext - IC Design Text vektorisieren (ICD)

Synopsis

```
int icd_vecttext(              // Status
      double;                  // X-Koordinate (STD2)
      double;                  // Y-Koordinate (STD2)
      double;                  // Drehwinkel (STD3)
      int [0,1];               // Spiegelung (STD14)
      double ]0.0,[;           // Text Größe (STD2)
      int [0,1];               // Physical Flag:
                               //    0 = Logical
                               //    1 = Physical
      int [0,2];               // Lagenspiegelung:
                               //    0 = Spiegelung aus
                               //    1 = X-Spiegelung
                               //    2 = Y-Spiegelung
      int [0,[;                // Text Stil
      string;                  // Text Zeichenkette
      * int;                   // Vektorisierungsfunktion
      );
```

Beschreibung

Die Funktion **icd_vecttext** vektorisiert den übergebenen Text unter Verwendung des aktuell geladenen Zeichensatzes. Dazu wird für jedes Textsegment die übergebene Vektorisierungsfunktion aufgerufen. Der Rückgabewert dieser Funktion ist ungleich Null, wenn ungültige Parameter angegeben wurden oder die vom Benutzer definierte Vektorisierungsfunktion einen Wert ungleich Null zurückgegeben hat.

Vektorisierungsfunktion

```
int vecfuncname(
      double x1,               // X-Koordinate erster Punkt (STD2)
      double y1,               // Y-Koordinate erster Punkt (STD2)
      double x2,               // X-Koordinate zweiter Punkt (STD2)
      double y2                // Y-Koordinate zweiter Punkt (STD2)
      )
{
      // Verarbeitungsprogramm
      :
      return(errstat);
}
```

Der Rückgabewert dieser Funktion sollte ungleich Null sein, wenn ein Fehler aufgetreten ist.

C.5.2 Chipeditor-Funktionen

Die nachfolgend aufgelisteten Systemfunktionen sind dem Aufruftyp CED zugeordnet, d.h. diese Funktionen können im **Chipeditor** aufgerufen werden:

ced_asklayer - CED Lagenauswahl (CED)

Synopsis

```
int ced_asklayer(               // Status
      & int;                    // Lagenrückgabe (ICD1)
      );
```

Beschreibung

Die Funktion **ced_asklayer** aktiviert im **Chipeditor** ein Lagenauswahlmenü. Der Rückgabewert ist Null bei erfolgter Lagenwahl oder (-1) bei Wahl des Menüpunktes Abbruch.

ced_delelem - CED Element löschen (CED)

Synopsis

```
int ced_delelem(                // Status
      & index I_FIGURE;         // Element
      );
```

Beschreibung

Die Funktion **ced_delelem** löscht das übergebene Element aus der Elementliste. Der Rückgabewert ist Null bei erfolgter Löschung und (-1), wenn das übergebene Element ungültig ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **I_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

Siehe auch

Funktion **ced_drawelem**.

ced_drawelem - CED Elementanzeige aktualisieren (CED)

Synopsis

```
void ced_drawelem(
      index I_FIGURE;           // Element
      int [0, 4];               // Zeichenmodus (STD19)
      );
```

Beschreibung

Die Funktion **ced_drawelem** aktualisiert die Anzeige des angegebenen Elements unter Verwendung des spezifizierten Zeichenmodus.

Siehe auch

Funktion **ced_delelem**.

ced_elemangchg - CED Elementwinkel ändern (CED)

Synopsis

```
int ced_elemangchg(             // Status
      & index I_FIGURE;         // Element
      double;                   // Neuer Winkel (STD3)
      );
```

Beschreibung

Die Funktion **ced_elemangchg** ändert den Drehwinkel des übergebenen Elements. Der Drehwinkel wird ausgehend vom Nullwinkel eingestellt, d.h. der vorhergehende Drehwinkel des Elements hat keinen Einfluss auf das Ergebnis. Die Winkelangabe wird als Bogenmaßwert interpretiert. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht drehbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **I_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ced_elemfixchg - CED Element fixiert-Flag ändern (CED)

Synopsis

```
int ced_elemfixchg(             // Status
      & index I_FIGURE;         // Element
      int [0,1];                // Neues fixiert Flag (STD11)
      );
```

Beschreibung

Die Funktion **ced_elemfixchg** ändert den Fixiert-Modus des übergebenen Elements. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht fixierbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **I_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ced_elemgrpchg - CED Element Gruppenflag ändern (CED)

Synopsis

```
int ced_elemgrpchg(             // Status
      index I_FIGURE;           // Element
      int [0,2];                // Neues Gruppenflag (STD13)
      );
```

Beschreibung

Die Funktion **ced_elemgrpchg** ändert die Gruppenzugehörigkeit des übergebenen Elements. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig ist oder (-2) wenn es sich nicht um ein gruppenselektierbares Element handelt. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

ced_elemlaychg - CED Elementlage ändern (CED)

Synopsis

```
int ced_elemlaychg(            // Status
      & index I_FIGURE;        // Element
      int;                     // Neue Lage (ICD1)
      );
```

Beschreibung

Die Funktion **ced_elemlaychg** ändert die Lagenzugehörigkeit des übergebenen Elements. Die Lage kann für Flächen, Leiterbahnen und Texte geändert werden. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn die Lage nicht änderbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **I_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ced_elemmirrchg - CED Elementspiegelung ändern (CED)

Synopsis

```
int ced_elemmirrchg(           // Status
      & index I_FIGURE;        // Element
      int [0,2];               // Neuer Spiegelungsmodus (STD14|ICD3)
      );
```

Beschreibung

Die Funktion **ced_elemmirrchg** ändert den Spiegelungsmodus des übergebenen Elements. Der Spiegelungsmodus kann bei Flächen, Texten, benannten und unbenannten Referenzen geändert werden. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig ist oder (-2) wenn es keinen Spiegelungsmodus besitzt. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **I_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ced_elemposchg - CED Elementposition ändern (CED)

Synopsis

```
int ced_elemposchg(            // Status
      & index I_FIGURE;        // Element
      double;                  // X-Position (STD2)
      double;                  // Y-Position (STD2)
      );
```

Beschreibung

Die Funktion **ced_elemposchg** ändert die Position des übergebenen Elements. Bei Flächen/Leiterbahnen wird die Fläche/Leiterbahn so verschoben, dass der erste Punkt der Fläche/Leiterbahn auf der angegebenen Position zu liegen kommt. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht positionierbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **I_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ced_elemsizechg - CED Elementgröße ändern (CED)

Synopsis

```
int ced_elemsizechg(            // Status
      & index I_FIGURE;         // Element
      double;                   // Neue Größe (STD2)
      );
```

Beschreibung

Die Funktion **ced_elemsizechg** ändert die Größe des übergebenen Elements. Bei Leiterbahnen wird mit der Größe die Leiterbahnbreite spezifiziert. Bei benannten und unbenannten Referenzen gibt die Größe den Skalierungsfaktor an. Eine Größenänderung ist nur bei Texten, Leiterbahnen, benannten und unbenannten Bauteilreferenzen möglich. Der Rückgabewert ist Null bei erfolgter Änderung, (-1) wenn das übergebene Element ungültig oder (-2) wenn es nicht größenveränderbar ist. Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **I_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ced_getlaydispmode - CED Lagenanzeigemodus abfragen (CED)

Synopsis

```
int ced_getlaydispmode(         // Anzeigemodus (ICD9)
      int;                      // Lage (ICD1)
      );
```

Beschreibung

Die Funktion **ced_getlaydispmode** ermittelt den Bildschirmanzeigemodus (ICD9) für die angegebene Lage.

ced_getmincon - CED Mincon-Funktion abfragen (CED)

Synopsis

```
int ced_getmincon(              // Mincon Funktionstyp (ICD10)
      );
```

Beschreibung

Der Rückgabewert der Funktion **ced_getmincon** entspricht dem im **Chipeditor** aktuell eingestellten Wert des Mincon-Modus für die Airlineanzeige (ICD10).

ced_getpathwidth - CED Bahnenstandardbreite abfragen (CED)

Synopsis

```
void ced_getpathwidth(
      & double;                 // Schmal Standardbreite (STD2)
      & double;                 // Breit Standardbreite (STD2)
      );
```

Beschreibung

Die Funktion **ced_getpathwidth** gibt in den beiden Parametern die Werte der aktuell im **Chipeditor** eingestellten Standardbreiten für schmale und breite Leiterbahnen zurück.

ced_getpickpreflay - CED Vorzugslage abfragen (CED)

Synopsis

```
int ced_getpickpreflay(       // Vorzugslage (ICD1)
      );
```

Beschreibung

Der Rückgabewert der Funktion **ced_getpickpreflay** entspricht der Vorzugslage (ICD1) für Elementwahl im **Chipeditor**.

ced_getwidedraw - CED Breitendarstellung abfragen (CED)

Synopsis

```
double ced_getwidedraw(       // Breite (STD2)
      );
```

Beschreibung

Der Rückgabewert der Funktion **ced_getwidedraw** entspricht der Breite, ab der im **Chipeditor** Leiterbahnen auf dem Bildschirm in Flächendarstellung angezeigt werden.

ced_groupselect - CED Gruppenselektion (CED)

Synopsis

```
int ced_groupselect(          // Anzahl Änderungen oder (-1) bei Fehler
      int [0,3];              // Element Selektionstyp:
                              //    0 = Selektion nach Elementtyp
                              //    1 = Selektion nach Lage
                              //    2 = Selektion nach Fixiertflag
                              //    3 = Selektion nach Sichtbarkeit
      int;                    // Element Selektionswert entspr. Selektionstyp:
                              //    0 - Elementtyp (0|ICD5)
                              //    1 - Elementlage (ICD1)
                              //    2 - Element-Fixiertflag (STD11)
                              //    3 - Elementsichtbarkeit (0|1)
      int [0,2];              // Neues Gruppenflag (STD13)
      );
```

Beschreibung

Die Funktion **ced_groupselect** ändert die Gruppenzugehörigkeit aller Element des spezifizierten Typs bzw. mit der spezifizierten Eigenschaft. Der Rückgabewert entspricht der Anzahl der durchgeführtem Änderungen oder dem Wert (-1) bei fehlerhaften bzw. inkompatiblen Parameterangaben. Der Selektionswert Null bei der Selektion nach dem Elementtyp kann dazu benutzt werden, Elemente *beliebigen* Typs auszuwählen.

Warnung

Interne **IC-Design**-Elementtypen wie z.B. die Standardvia-Definition(en) sind von der Gruppen(de)selektion mit **ced_groupselect** ausgenommen, um ein versehentliches Löschen bzw. Ändern derartiger Elemente durch die anschließende Anwendung anderer Gruppenfunktionen zu verhindern.

ced_highlnet - CED Highlight Netz ein/aus (CED)

Synopsis

```
int ced_highlnet(             // Status
      int [0,[;               // Netznummer
      int [0,1];              // Highlightmodus (0 = aus, 1 = ein)
      );
```

Beschreibung

Die Funktion **ced_highlnet** setzt den Highlightmodus des Netzes mit der übergebenen Netznummer. Es wird ein Wert ungleich Null zurückgegeben, wenn ein ungültiger Highlightmodus oder eine ungültige Netznummer angegeben wurde.

ced_layergrpchg - CED Gruppenselektion nach Lage (CED)

Synopsis

```
int ced_layergrpchg(           // Anzahl Elemente
      int;                     // Lagennummer (ICD1)
      int [0,2];               // Neues Gruppenflag (STD13)
      );
```

Beschreibung

Die Funktion **ced_layergrpchg** ändert die Gruppenzugehörigkeit aller Elemente, die auf der angegebenen Lage platziert sind. Der Rückgabewert ist die Anzahl der (de)selektierten Elemente oder (-1) bei Fehler.

ced_partaltmacro - CED Bauteilzellentyp ändern (CED)

Synopsis

```
int ced_partaltmacro(          // Status
      string;                  // Bauteilname
      string;                  // Neuer Zellen-Bibliotheksteilname
      );
```

Beschreibung

Die Funktion **ced_partaltmacro** ändert den Zellentyp des angegebenen Bauteiles. Ein Rückgabewert von Null zeigt eine erfolgreiche Änderung an. Bei ungültigen Eingabedaten wird (-1) zurückgegeben, (-2) wenn der neue Zellentyp nicht alle für dieses Bauteil in der Netzliste verwendeten Pins enthält (Zellenänderung wird trotzdem durchgeführt), (-3) wenn das Bauteil nicht in der Netzliste vorhanden ist, (-4) wenn der neue Zellentyp für dieses Bauteil nicht erlaubt ist, (-5) wenn der neue Zellentyp nicht ladbar ist, (-6) wenn die Zellendaten nicht in die Jobdatenbank kopiert werden konnten und (-7) wenn versucht wurde, in einem Programmlauf einen Bauteilzellentyp mehrfach umzuändern (z.B. **a** in **b** und anschließend **b** in **c**). Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion sollte nicht innerhalb von **I_CPART**-Index-Schleifen aufgerufen werden, da die vor dem Aufruf der Funktion belegten **I_CPART**-Indexvariablen anschließend ungültig sind.

ced_partnamechg - CED Bauteilname in Netzliste ändern (CED)

Synopsis

```
int ced_partnamechg(           // Status
      string;                  // Alter Bauteilname
      string;                  // Neuer Bauteilname
      );
```

Beschreibung

Die Funktion **ced_partnamechg** ändert den Namen des angegebenen Bauteiles in der Netzliste. Ein Rückgabewert von Null zeigt eine erfolgreiche Änderung an. Bei ungültigen Eingabedaten wird (-1) zurückgegeben, (-2) wenn das Bauteil nicht platziert ist, (-3) wenn das Bauteil nicht in der Netzliste vorhanden ist, (-4) wenn der neue Name schon definiert ist und (-5) wenn versucht wurde, in einem Programmlauf ein Bauteil mehrfach umzubenennen (z.B. **a** in **b** und anschließend **b** in **c**). Die Änderung kann nach dem Programmlauf mit Undo wieder rückgängig gemacht werden.

Warnung

Diese Funktion verändert die Netzliste und erfordert daher einen anschließenden Backannotation-Lauf. Die Funktion sollte auch nicht innerhalb von **I_CPART**-Index-Schleifen aufgerufen werden, da die vor dem Aufruf der Funktion belegten **I_CPART**-Indexvariablen anschließend ungültig sind.

ced_pickelem - CED Element selektieren (CED)

Synopsis

```
int ced_pickelem(                // Status
      & index I_FIGURE;          // Rückgabe Element
      int [1,8];                 // Elementtyp (ICD5 außer 6)
      );
```

Beschreibung

Mit der Funktion **ced_pickelem** kann vom Benutzer mit der Maus ein Element des gewünschten übergebenen Typs selektiert werden. Der Rückgabewert ist Null bei erfolgter Selektion und (-1) wenn an der Pickposition kein Element des gewünschten Typs gefunden wurde.

ced_setlaydispmode - CED Lagenanzeigemodus setzen (CED)

Synopsis

```
int ced_setlaydispmode(          // Status
      int [0,99];                // Lage (ICD1)
      int [0,127];               // Anzeigemodus (ICD9)
      );
```

Beschreibung

Die Funktion **ced_setlaydispmode** setzt den Bildschirmanzeigemodus für die angegebenen Lage. Der Rückgabewert ist ungleich Null, wenn der Anzeigemodus nicht gesetzt werden konnte.

ced_setmincon - CED Mincon-Funktion setzen (CED)

Synopsis

```
int ced_setmincon(               // Status
      int [0,8];                 // Mincon Funktionstyp (ICD10)
      );
```

Beschreibung

Die Funktion **ced_setmincon** setzt den Mincon-Modus für die Airlineanzeige im **Chipeditor**. Es wird ein Wert ungleich Null zurückgegeben, wenn ein ungültiger Modus angegeben wurde.

ced_setpathwidth - CED Bahnenstandardbreiten setzen (CED)

Synopsis

```
int ced_setpathwidth(            // Status
      double ]0.0,[;             // Schmal Standardbreite (STD2)
      double ]0.0,[;             // Breit Standardbreite (STD2)
      );
```

Beschreibung

Die Funktion **ced_setpathwidth** setzt die Standardbreiten für schmale und breite Leiterbahnen im **Chipeditor**. Es wird ein Wert ungleich Null zurückgegeben, wenn ungültige Breiten spezifiziert wurden.

ced_setpickpreflay - CED Vorzugslage setzen (CED)

Synopsis

```
int ced_setpickpreflay(          // Status
      int;                       // Lage (ICD1)
      );
```

Beschreibung

Die Funktion **ced_setpickpreflay** setzt die Vorzugslage für Elementwahl im **Chipeditor**. Es wird ein Wert ungleich Null zurückgegeben, wenn keine gültige Lage angegeben wurde.

ced_setwidedraw - CED Breitendarstellung setzen (CED)

Synopsis

```
int ced_setwidedraw(            // Status
      double ]0.0,[;            // Breite (STD2)
      );
```

Beschreibung

Die Funktion **ced_setwidedraw** setzt die Breite, ab der im **Chipeditor** Leiterbahnen auf dem Bildschirm in Flächendarstellung angezeigt werden. Es wird ein Wert ungleich Null zurückgegeben, wenn eine ungültige Breite spezifiziert wurde.

ced_storepart - CED Bauteil platzieren (CED)

Synopsis

```
int ced_storepart(              // Status
      string;                   // Bauteilname
      string;                   // Bauteil Bibiotheksteilname
      double;                   // X-Koordinate (STD2)
      double;                   // Y-Koordinate (STD2)
      double;                   // Drehwinkel (STD3)
      double;                   // Skalierungsfaktor
      int [0,1];                // Spiegelungsmodus (STD14)
      );
```

Beschreibung

Die Funktion **ced_storepart** platziert ein Bauteil mit den angegebenen Parametern auf dem gegenwärtig geladenen Chip Layoutelement. Wird eine Leerzeichenkette für den Bauteilnamen übergeben, so wird das nächste unplatzierte Bauteil der Netzliste verwendet. Der Rückgabewert ist gleich Null, wenn das Bauteil erfolgreich platziert wurde, (-1) bei ungültigen Daten, (-2) wenn alle Bauteile bereits platziert sind, (-3) wenn das Bauteil schon platziert ist, (-4) wenn es nicht ladbar ist, (-5) wenn die Bauteilpins nicht mit der Netzliste übereinstimmen und (-6) wenn die Bauteildaten nicht in die Jobdatenbank kopiert werden konnten.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **I_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ced_storepath - CED Bahn platzieren (CED)

Synopsis

```
int ced_storepath(              // Status
      int [0,99];               // Lage (ICD1)
      double ]0.0,[;            // Bahnbreite (STD2)
      );
```

Beschreibung

Die Funktion **ced_storepath** erzeugt aus der internen Punktliste unter Verwendung der angegebenen Parameter eine Leiterbahn auf dem gegenwärtig geladenen Chip Layoutelement. Der Rückgabewert ist gleich Null, wenn die Bahn erfolgreich platziert wurde, (-1) wenn kein gültiges Element geladen ist, (-2) bei ungültigen Parametern, oder (-3) wenn die Punktliste ungültig ist.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **I_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ced_storepoly - CED Fläche platzieren (CED)

Synopsis

```
int ced_storepoly(              // Status
      int;                      // Lage (ICD1)
      int [1,4];                // Polygontyp (ICD4)
      int [0,2];                // Spiegelungsmodus (ICD3)
      );
```

Beschreibung

Die Funktion **ced_storepoly** generiert aus der mit **bae_storepoint** erzeugten internen Punktliste unter Verwendung der angegebenen Parameter eine Fläche auf dem gegenwärtig geladenen IC-Layoutelement. Der Rückgabewert ist gleich Null, wenn die Fläche erfolgreich platziert wurde, (-1) wenn kein gültiges Element geladen ist, (-2) bei ungültigen Parametern, oder (-3) wenn die Punktliste für den gegebenen Flächentyp ungültig ist.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **I_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.

ced_storetext - CED Text platzieren (CED)

Synopsis

```
int ced_storetext(              // Status
      string;                   // Textzeichenkette
      double;                   // X-Koordinate (STD2)
      double;                   // Y-Koordinate (STD2)
      double;                   // Drehwinkel (STD3)
      double ]0.0,[;            // Text Größe (STD2)
      int;                      // Lage (ICD1)
      int [0,1];                // Spiegelungsmodus (STD14)
      );
```

Beschreibung

Die Funktion **ced_storetext** platziert einen Text mit den angegebenen Parametern auf dem gegenwärtig geladenen Chip Layoutelement. Der Rückgabewert ist ungleich Null, wenn ungültige Daten übergeben wurden.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **I_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden. Es können maximal 40 Zeichen der übergebenen Textzeichenkette gespeichert werden. Bei Übergabe längerer Zeichenketten gibt die Funktionen den Fehlerstatus zur Kennzeichnung ungültiger Parameter zurück.

ced_storeuref - CED Via bzw. Subbauteil platzieren (CED)

Synopsis

```
int ced_storeuref(            // Status
      string;                 // Referenz Bibiotheksteilname
      double;                 // X-Koordinate (STD2)
      double;                 // Y-Koordinate (STD2)
      double;                 // Drehwinkel (STD3)
      double;                 // Skalierungsfaktor
      int [0,1];              // Spiegelung (STD14) (für Subbauteile)
      );
```

Beschreibung

Die Funktion **ced_storeuref** platziert eine namenlose Referenz mit den angegebenen Parametern auf dem gegenwärtig geladenen Layoutelement. Namenlose Referenzen sind die Vias auf Chip Layoutebene und die Subbauteile auf Bauteilebene. Spiegelung, Skalierung und Drehwinkel werden für Vias ignoriert. Der Rückgabewert ist gleich Null, wenn die Referenz erfolgreich platziert wurde, (-1) bei ungültigen Daten, (-2) wenn sie nicht ladbar ist und (-3) wenn die Referenzdaten nicht in die Jobdatenbank kopiert werden konnten.

Warnung

Diese Funktion ändert die aktuelle Figurenliste und sollte daher mit Vorsicht in `forall`-Schleifen zur Iteration von **I_FIGURE**-Indexvariablen verwendet werden, um undefinierte Ergebnisse beim Zugriff auf die Figurenliste bzw. Endlosschleifen zu vermeiden.